

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2022, December 18–20, 2022, IIT Madras, Chennai,
India

Edited by

Anuj Dawar

Venkatesan Guruswami



Editors

Anuj Dawar 

University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

Venkatesan Guruswami 

University of California, Berkeley, USA
venkatg@berkeley.edu

ACM Classification 2012

Theory of computation; Computing methodologies; Software and its engineering

ISBN 978-3-95977-261-7

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/978-3-95977-261-7>.

Publication date

December, 2022

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>.

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): <https://creativecommons.org/licenses/by/4.0/legalcode>.



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2022.0

ISBN 978-3-95977-261-7

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.

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Anuj Dawar and Venkatesan Guruswami</i>	0:ix
Program Committee	
.....	0:xi
List of External Reviewers: Track A	
.....	0:xiii
List of External Reviewers: Track B	
.....	0:xv

Invited Talks

Algorithms for Uncertain Environments: Going Beyond the Worst-Case	
<i>Anupam Gupta</i>	1:1–1:1
Why MCSP Is a More Important Problem Than SAT	
<i>Rahul Santhanam</i>	2:1–2:1
The True Colors of Memory: A Tour of Chromatic-Memory Strategies in Zero-Sum Games on Graphs	
<i>Patricia Bouyer, Mickael Randour, and Pierre Vandenholve</i>	3:1–3:18
Expanders in Higher Dimensions	
<i>Irit Dinur</i>	4:1–4:1

Regular Papers

Packing Arc-Disjoint 4-Cycles in Oriented Graphs	
<i>Jasine Babu, R. Krithika, and Deepak Rajendraprasad</i>	5:1–5:16
Approximate Representation of Symmetric Submodular Functions via Hypergraph Cut Functions	
<i>Calvin Beideman, Karthekeyan Chandrasekaran, Chandra Chekuri, and Chao Xu</i>	6:1–6:18
The DAG Visit Approach for Pebbling and I/O Lower Bounds	
<i>Gianfranco Bilardi and Lorenzo De Stefani</i>	7:1–7:23
Counting and Sampling from Substructures Using Linear Algebraic Queries	
<i>Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Manaswi Paraashar</i>	8:1–8:20
Derandomization via Symmetric Polytopes: Poly-Time Factorization of Certain Sparse Polynomials	
<i>Pranav Bisht and Nitin Saxena</i>	9:1–9:19
On Solving Sparse Polynomial Factorization Related Problems	
<i>Pranav Bisht and Ilya Volkovich</i>	10:1–10:22

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami



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

Complexity of Spatial Games <i>Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Ismaël Jecker, and Jakub Svoboda</i>	11:1–11:14
Robustly Separating the Arithmetic Monotone Hierarchy via Graph Inner-Product <i>Arkadev Chattopadhyay, Utsab Ghosal, and Partha Mukhopadhyay</i>	12:1–12:20
Inscribing or Circumscribing a Histogram to a Convex Polygon <i>Jaehoon Chung, Sang Won Bae, Chan-Su Shin, Sang Duk Yoon, and Hee-Kap Ahn</i>	13:1–13:16
More Verifier Efficient Interactive Protocols for Bounded Space <i>Joshua Cook</i>	14:1–14:18
Improved Quantum Query Upper Bounds Based on Classical Decision Trees <i>Arjan Cornelissen, Nikhil S. Mande, and Subhasree Patro</i>	15:1–15:22
On the VNP-Hardness of Some Monomial Symmetric Polynomials <i>Radu Curticapean, Nutan Limaye, and Srikanth Srinivasan</i>	16:1–16:14
Online Piercing of Geometric Objects <i>Minati De, Saksham Jain, Sarat Varma Kallepalli, and Satyam Singh</i>	17:1–17:16
Half-Guarding Weakly-Visible Polygons and Terrains <i>Nandhana Duraisamy, Hannah Miller Hillberg, Ramesh K. Jallu, Erik Krohn, Anil Maheshwari, Subhas C. Nandy, and Alex Pahlow</i>	18:1–18:17
A Structural and Algorithmic Study of Stable Matching Lattices of “Nearby” Instances, with Applications <i>Rohith Reddy Gangam, Tung Mai, Nitya Raju, and Vijay V. Vazirani</i>	19:1–19:20
Degree-Restricted Strength Decompositions and Algebraic Branching Programs <i>Fulvio Gesmundo, Purnata Ghosal, Christian Ikenmeyer, and Vladimir Lysikov</i>	20:1–20:15
A Simple Polynomial Time Algorithm for Max Cut on Laminar Geometric Intersection Graphs <i>Utkarsh Joshi, Saladi Rahul, and Jossen Joe Thoppil</i>	21:1–21:12
Stable Matchings with One-Sided Ties and Approximate Popularity <i>Telikepalli Kavitha</i>	22:1–22:17
Geometry Meets Vectors: Approximation Algorithms for Multidimensional Packing <i>Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas</i>	23:1–23:22
Black Box Absolute Reconstruction for Sums of Powers of Linear Forms <i>Pascal Koiran and Subhayan Saha</i>	24:1–24:17
When You Come at the King You Best Not Miss <i>Oded Lachish, Felix Reidl, and Chhaya Trehan</i>	25:1–25:12
Complexity of Fault Tolerant Query Complexity <i>Ramita Maharjan and Thomas Watson</i>	26:1–26:11
Romeo and Juliet Meeting in Forest like Regions <i>Neeldhara Misra, Manas Mulpuri, Prafullkumar Tale, and Gaurav Viramgami</i>	27:1–27:22

New Characterizations of Core Imputations of Matching and b -Matching Games <i>Vijay V. Vazirani</i>	28:1–28:13
Algorithms and Hardness Results for Computing Cores of Markov Chains <i>Ali Ahmadi, Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, Roodabeh Safavi, and Đorđe Žikelić</i>	29:1–29:20
Computing Threshold Budgets in Discrete-Bidding Games <i>Guy Avni and Suman Sadhukhan</i>	30:1–30:18
Dependency Matrices for Multiplayer Strategic Dependencies <i>Dylan Bellier, Sophie Pinchinat, and François Schwarzentruber</i>	31:1–31:21
Semilinear Representations for Series-Parallel Atomic Congestion Games <i>Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur</i>	32:1–32:20
Playing (Almost-)Optimally in Concurrent Büchi and Co-Büchi Games <i>Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux</i>	33:1–33:18
Ambiguity Through the Lens of Measure Theory <i>Olivier Carton</i>	34:1–34:14
Phase Semantics for Linear Logic with Least and Greatest Fixed Points <i>Abhishek De, Farzad Jafarrahmani, and Alexis Saurin</i>	35:1–35:23
Natural Colors of Infinite Words <i>Rüdiger Ehlers and Sven Schewe</i>	36:1–36:17
Synthesizing Dominant Strategies for Liveness <i>Bernd Finkbeiner and Noemi Passing</i>	37:1–37:19
Low-Latency Sliding Window Algorithms for Formal Languages <i>Moses Ganardi, Louis Jachiet, Markus Lohrey, and Thomas Schwentick</i>	38:1–38:23
The Design and Regulation of Exchanges: A Formal Approach <i>Mohit Garg and Suneel Sarswat</i>	39:1–39:21
Parikh Automata over Infinite Words <i>Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann</i>	40:1–40:20
New Analytic Techniques for Proving the Inherent Ambiguity of Context-Free Languages <i>Florent Koechlin</i>	41:1–41:22
Synthesis of Privacy-Preserving Systems <i>Orna Kupferman and Ofer Leshkowitz</i>	42:1–42:23
A Generic Polynomial Time Approach to Separation by First-Order Logic Without Quantifier Alternation <i>Thomas Place and Marc Zeitoun</i>	43:1–43:22
A Technique to Speed up Symmetric Attractor-Based Algorithms for Parity Games <i>K. S. Thejaswini, Pierre Ohlmann, and Marcin Jurdziński</i>	44:1–44:20

■ Preface

This volume contains the proceedings of the 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022). The conference was held on December 18–21, 2022 on the campus of IIT Madras in Chennai, India. This was the first time in three years that it was possible for the conference to be held in person. Some participants were nonetheless able to participate remotely.

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 (Venkatesan Guruswami for Track A and Anuj Dawar 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 comprised of 5 invited talks, 24 contributed talks in Track A, and 16 in Track B. This volume contains all the contributed papers from the two tracks, short abstracts of three of the invited talks as well as a longer paper accompanying one of the invited talks. We thank all the authors who submitted their papers to FSTTCS 2022. 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: Patricia Bouyer-Decitre (CNRS and ENS Paris-Saclay), Irit Dinur (Weizmann Institute), Anupam Gupta (Carnegie Mellon University), Akash Lal (Microsoft Research), and Rahul Santhanam (University of Oxford).

The main conference was accompanied by three workshops or other colocated events: *Algorithms under Uncertainty* (organized by Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar and Debmalya Panigrahi), *Fine-grained Cryptography* (organized by Divesh Aggarwal, Huck Bennett, Alexander Golovnev, Rajendra Kumar and Noah Stephens-Davidowitz), and *SAT+SMT Winter School* (organized by Supratik Chakraborty, Ashutosh Gupta, Saurabh Joshi, Kumar Madhukar, Kuldeep S. Meel and Subodh Sharma).

We are indebted to the organizing committee members: Meghana Nasre (IIT Madras), Chandrashekhara Sahasrabudhe (ACM India, Member), and Jayalal Sarma (IIT Madras) so ably supported by members of the Theory Group and CCD group at IIT Madras. They made all the necessary arrangements for the conference. We thank S.P. Suresh (CMI, Chennai) for maintaining the conference web page. We thank Michael Wagner and the friendly staff at Dagstuhl LIPIcs for helping us put together the proceedings. Finally, we thank the members of the Steering Committee, especially Amit Kumar, and the PC Chairs from FSTTCS 2021 (Chandra Chekuri and Mikołaj Bojańczyk), for providing pertinent information and advice about various aspects of the conference.

Anuj Dawar and Venkatesan Guruswami
October 2022



■ Program Committee

Track A

- Venkatesan Guruswami (University of California, Berkeley) – Co-chair
- V. Arvind (The Institute of Mathematical Sciences)
- Divesh Aggarwal (National University of Singapore)
- Diptarka Chakraborty (National University of Singapore)
- Radu Curticapean (IT University of Copenhagen)
- Sumegha Garg (Harvard University)
- Badih Ghazi (Google Research)
- Tom Gur (University of Warwick)
- Euiwoong Lee (University of Michigan)
- Debmalya Panigrahi (Duke University)
- Pravesh Kothari (Carnegie Mellon University)
- Nicole Megow (Universität Bremen)
- Prajakta Nimbhorkar (Chennai Mathematical Institute)
- Rishi Saket (Google Research, India)
- Ramprasad Satharishi (Tata Institute of Fundamental Research)
- Rakesh Venkat (Indian Institute of Technology, Hyderabad)
- David Wajc (Stanford University)
- Amir Yehudayoff (Technion, Israel)
- Meirav Zahavi (Ben-Gurion University, Israel)

Track B

- Anuj Dawar (University of Cambridge) – Co-chair
- Ashutosh Trivedi (University of Colorado Boulder)
- Deepak D'Souza (Indian Institute of Science)
- Igor Walukiewicz (Université de Bordeaux, France)
- Jonni Virtema (University of Sheffield, UK)
- Laura Bozzelli (University of Naples, Italy)
- Laure Daviaud (City University of London)
- Maribel Fernandez (King's College London)
- Mohamed Faouzi Atig (Uppsala University, Sweden)
- Natasha Alechina (Utrecht University, Netherlands)
- Nathan Lhote (Aix-Marseille University, France)
- Olaf Beyersdorff (Friedrich Schiller University Jena, Germany)
- Radha Jagadeesan (DePaul University)
- Shaul Almagor (Technion, Israel)
- Shibashis Guha (Tata Institute of Fundamental Research)
- Supratik Chakraborty (Indian Institute of Technology, Bombay)



■ List of External Reviewers: Track A

Aditya Anand	Akanksha Agrawal
Akshayaram Srinivasan	Alex Steiger
Arantxa Zapico	Ben Lee Volk
Boaz Patt-Shamir	Bodhayan Roy
Chandan Saha	Chetan Gupta
Christoph Grunau	Christopher Liaw
Csaba Toth	Daniel Schmand
Eldon Chung	Erin Taylor
Fahad Panolan	Fedor Fomin
Geeverghese Philip	Girija Limaye
Gregory Rosenthal	Gunjan Kumar
Igor Zlotchi	Ilan Cohen
Jayalal Sarma	Jiong Guo
Jocelyn Thiebaut	Kai Jin
Klara Nosan	Kushagra Chatterjee
Makrand Sinha	Matthias Mnich
Matthieu Perrin	Mingyu Xiao
Mrinal Kumar	Nai-Hui Chia
Nathaniel Kell	Neeldhara Misra
Oded Lachish	Omrit Filtser
Parth Mittal	Partha Mukhopadhyay
Petr Golovach	Piyush Srivastava
Pranjal Dutta	Pratik Ghosal
Prerona Chatterjee	Pushkar Joglekar
Rajendra Kumar	Rob van Stee
Rogers Mathew	Ruoxu Cen
Sheikh Shakil Akhtar	Siddhartha Jain
Srijita Kundu	Srikanth Srinivasan
Srinivasa Rao Satti	Suprovat Ghoshal
Suthee Ruangwises	Tomas Peitl
Ton Kloks	Venkatesh Raman
Vishwas Bhargava	Yuri Faenza
Zeyong Li	



■ List of External Reviewers: Track B

Adriano Peron	Antonio Casares
Asaf Yeshurun	Ashutosh Gupta
B. Srivathsan	Benjamin Böhm
Bernd Finkbeiner	C. Aiswarya
Cesar Sanchez	Corto Mascle
Dario Della Monica	Dror Fried
Geraud Senizergues	Hoang Nga Nguyen
Hrishikesh Karmarkar	Jakub Gajarsky
Jean-Francois Raskin	K. S. Thejaswini
Laurent Doyen	Luc Dartois
Luc Spachmann	M. Praveen
Marc Schroder	Markus L. Schmid
Martin Zimmermann	Massimo Benerecetti
Michael Cadilhac	Mukesh Tiwari
Patrick Totzke	Paweł Parys
Pietro Sala	Prakash Saivasan
Rémi Morvan	S. Akshay
Stanly John Samuel	Sumanth Prabhu
Tim Hoffmann	Tomas Masopust
Tuomas Hakoniemi	Vrunda Dave
Yuta Takahashi	



Algorithms for Uncertain Environments: Going Beyond the Worst-Case

Anupam Gupta ✉

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Analyzing the performance of algorithms in both the worst case and the average case are cornerstones of computer science: these are two different ways to understand how well algorithms perform. Over the past two decades, there has been a concerted effort to understand the performance of algorithms in models that go beyond these two extremes. In this talk I will discuss some of the proposed models and approaches, particularly for problems related to online algorithms, where decisions must be made sequentially without knowing future portions of the input.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Optimization under Uncertainty, Online Algorithms, Beyond Worst Case Analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.1

Category Invited Talk



© Anupam Gupta;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).




Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 1; pp. 1:1–1:1



Leibniz International Proceedings in Informatics

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

Why MCSP Is a More Important Problem Than SAT

Rahul Santhanam   

Department of Computer Science, University of Oxford, UK

Abstract

CNF Satisfiability (SAT) and its variants are generally considered the central problems in complexity theory, due to their applications in the theory of NP-completeness, logic, verification, probabilistically checkable proofs and parameterized complexity, among other areas. We challenge this conventional wisdom and argue that analysing the Minimum Circuit Size Problem (MCSP) and its relatives is more important from the perspective of fundamental problems in complexity theory, such as complexity lower bounds, minimal assumptions for cryptography, a robust theory of average-case complexity, and optimal results in hardness of approximation.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Minimum Circuit Size Problem, Satisfiability, Cryptography, Learning, Approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.2

Category Invited Talk

Funding Partially funded by EPSRC New Horizons Grant EP/V048201/1.



© Rahul Santhanam;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 2; pp. 2:1–2:1



Leibniz International Proceedings in Informatics

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

The True Colors of Memory: A Tour of Chromatic-Memory Strategies in Zero-Sum Games on Graphs

Patricia Bouyer 

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Mickael Randour 

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Pierre Vandenhove 

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Abstract

Two-player turn-based zero-sum games on (finite or infinite) graphs are a central framework in theoretical computer science – notably as a tool for controller synthesis, but also due to their connection with logic and automata theory. A crucial challenge in the field is to understand *how complex* strategies need to be to play optimally, given a type of game and a winning objective. In this invited contribution, we give a tour of recent advances aiming to characterize games where finite-memory strategies suffice (i.e., using a limited amount of information about the past). We mostly focus on so-called chromatic memory, which is limited to using colors – the basic building blocks of objectives – seen along a play to update itself. Chromatic memory has the advantage of being usable in different game graphs, and the corresponding class of strategies turns out to be of great interest to both the practical and the theoretical sides.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases two-player games on graphs, finite-memory strategies, chromatic memory, parity automata, ω -regularity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.3

Category Invited Talk

Funding This work has been partially supported by the ANR Project MAVeriQ (ANR-20-CE25-0012), and the F.R.S.-FNRS Grant n° F.4520.18 (ManySynth). Mickael Randour is an F.R.S.-FNRS Research Associate and a member of the TRAIL Institute. Pierre Vandenhove is an F.R.S.-FNRS Research Fellow.

Acknowledgements Most of the results presented here [4, 6, 7, 3] are related to the F.R.S.-FNRS project FrontieRS, led by the authors. Some of these were obtained in collaboration with Antonio Casares, Stéphane Le Roux, and Youssouf Oualhadj. We express our utmost gratitude to our delightful co-authors.

1 Introduction

Two-player turn-based zero-sum games on graphs. We consider games between two players, \mathcal{P}_1 and \mathcal{P}_2 , that are played on a (finite or infinite) graph, often called *arena*, whose set of vertices is partitioned into vertices controlled by \mathcal{P}_1 and vertices controlled by \mathcal{P}_2 . The players interact by moving a pebble from vertex to vertex, ad infinitum, following edges of the graph. The game starts in a given vertex, and the owner of the current vertex decides where to send the pebble next. The infinite path thereby created is called a *play*.



© Patricia Bouyer, Mickael Randour, and Pierre Vandenhove;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 3; pp. 3:1–3:18



Leibniz International Proceedings in Informatics

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

We assume that the edges of the graph are labeled with *colors* from a finite or infinite set. These colors are used to define the *objective* of the game: an objective is simply a language of infinite color sequences [24]. This general view of objectives encompasses all classical notions from the literature, qualitative and quantitative objectives alike.

The goal of \mathcal{P}_1 is to create a play whose projection to colors belongs to the objective whereas \mathcal{P}_2 tries to prevent it; hence our games are *zero-sum*. They are also *turn-based* as the players take turns moving the pebble depending on the owner of the current vertex. These moves are chosen according to the *strategy* of the player, which, in general, might use memory (bounded or not) of the past moves to prescribe the next action.

This type of games has been studied for decades, for a plethora of objectives: see many examples in [4]. Interestingly, virtually all such games (i.e., for all reasonable objectives) are known to be *determined* since Martin’s seminal result on Borel determinacy [36]. This means that for every vertex v , either \mathcal{P}_1 has a strategy that guarantees victory when the game starts in v , or \mathcal{P}_2 has one. Two natural questions follow: given a game, can we decide from which vertex each player can win, and what kind of strategy do they need to use?

Reactive synthesis. This survey focuses on the latter question, which is particularly relevant in the context of *controller synthesis* for reactive systems [25, 41, 8, 2]. This formal methods approach aims to automatically synthesize a provably-correct controller for a reactive system that operates within an uncontrollable environment. Through the game-theoretic metaphor, one can model the interaction between the system and its (possibly antagonistic) environment as a two-player zero-sum game on a graph: vertices of the graph model states of the system-environment pair whereas the specification of the system is encoded as a winning objective.

The goal of synthesis is to decide if \mathcal{P}_1 (the system) has a *winning strategy*, i.e., one that ensures the objective against all possible strategies of \mathcal{P}_2 (the environment), and to build such a strategy if it exists. Winning strategies are essentially formal blueprints for controllers to implement in practical applications: these will thus be correct by design.

A wide variety of objectives (and combinations thereof – e.g., [14, 33]) have been studied in the literature, notably to offer appropriate modeling power for applications in reactive synthesis. All can be expressed through the formalism of colors used in this paper.

Strategy complexity. Keeping in mind that strategies are used as blueprints for real-world controllers, one easily understands why their complexity is of utmost importance: the simpler the strategy, the easier and cheaper it will be to synthesize the corresponding controller and maintain it. On the theoretical level, comprehending which classes of strategies suffice to *play optimally* (i.e., win whenever winning is possible) for various classes of games is also a worthy venture as it may lead to more efficient solving algorithms and the identification of common grounds between these different classes of games.

Many classical objectives are known to be *memoryless-determined*: memoryless strategies, i.e., only using the current vertex as the basis for their decisions, suffice to play optimally. It is for example the case of mean-payoff (in finite graphs) [19] or parity (in finite and infinite graphs) [20, 45]. Yet, over the last decade, the need to model increasingly complex specifications has geared research toward games with more intricate objectives (e.g., [9, 5]) or objectives arising from the combination of simple ones (e.g., [33, 10]). When considering such rich objectives, memoryless strategies usually do not suffice, and one has to use strategies relying on a memory structure, which can be finite or infinite. A natural follow-up question is thus to quantify the memory that is needed for a given objective.

Two flavors of memory. Two models of strategies coexist in the recent literature. In both, a finite-memory strategy can be seen as the association of a memory structure – we will call this a *memory skeleton* – taking the form of a finite automaton and of a memoryless strategy defined, not on the arena, but on the product of the arena and this memory structure – this memoryless strategy is the *next-action* function in the classical Mealy machine model. The only but crucial difference is *how* the memory structure updates its state when an edge is taken in the arena.

In *chromatic memory* (e.g., [4, 7, 11]), the update function only considers the *color* of the edge, whereas in *chaotic memory* (e.g., [18, 31, 13]), the updates consider the actual edge of the arena. That is, in chaotic memory, two different edges bearing the same color may lead to different updates whereas this is not possible in chromatic memory. As such, chromatic memory can be seen as a restricted class of finite-memory strategies. Yet, interestingly, chromatic memory proves sufficient in most cases, and appears more robust with regard to general characterizations. An important feature of chromatic memory is that it allows to define a memory skeleton for an objective independently of an arena, which is impossible for chaotic memory as it explicitly uses the underlying arena. Indeed, chromatic memory coincides with the model used for *arena-independent* strategies in [4]. That being said, given a particular arena, chaotic memory may lead to smaller memory structures – state-wise, not necessarily true when counting transitions – as it can use additional knowledge of the graph; see examples in [11, 13, 32].

General characterizations vs. tight memory bounds. In this invited contribution, we survey recent advances in the study of chromatic-memory strategies, most stemming from a series of co-authored papers on the topic. One can identify two orthogonal yet complementary directions in our work.

Our first line of research aims to establish general characterizations for large classes of games. A typical example is our characterization of objectives for which arena-independent (chromatic) finite-memory strategies suffice [4] (in finite arenas), providing a finite-memory equivalent to Gimbert and Zielonka’s seminal result [24]. The philosophy of this research direction is to identify the common grounds between various objectives and to pinpoint the underlying source of complexity, going beyond the use of ad-hoc proofs and techniques. In addition to its fundamental interest, this endeavor permits to establish amenable criteria that one can check to establish that finite-memory optimal strategies exist in a wide range of contexts. Of course, due to its generality, this approach might not lead to perfectly tight memory bounds in particular contexts.

Our second topic of interest can be seen as an answer to this limitation, as it aims to provide tight (lower and upper) memory bounds for specific objective classes. We focus on ω -regular objectives and rely on their representations through different classes of automata in our approach.

Outline. The structure of our paper follows these two lines of research. In Section 2, we introduce the main concepts and notations. Section 3 surveys our results on general characterizations, mainly [4] for finite arenas and [7] for infinite ones. Section 4 focuses on tight bounds for specific ω -regular objectives – we notably present the results of [3]. Finally, Section 5 discusses open questions and future work.

We highlight that this paper is meant as an introductory survey to the topic and, as such, does not give a fully-detailed presentation of all concepts and results. We settled on a high-level, hopefully intuitive, exposition of the field, trying to highlight the interest and limits of our current knowledge, along with connections between the different results. Interested readers may find all details in the corresponding full papers.

Additional related work. This paper only considers *deterministic* games and the corresponding results. In general, these do not carry over to stochastic (one-player or two-player) games, and specific techniques are needed, both to establish results on memory, but also to study the need for randomness – another aspect of the complexity of strategies arising in this context. We mention some references on the topic: [28, 27, 17, 6, 34].

2 Games on Graphs, Objectives, Chromatic Memory

In the whole article, letter C refers to a (finite or infinite) non-empty set of *colors*. Given a set A , we write respectively A^* , A^+ , and A^ω for the set of finite, non-empty finite, and infinite sequences of elements of A . We denote by ε the empty word.

Arenas. We consider two players \mathcal{P}_1 and \mathcal{P}_2 . An *arena* is a tuple $\mathcal{A} = (V, V_1, V_2, E)$ such that V is a non-empty set of *vertices* and is the disjoint union of V_1 and V_2 , and $E \subseteq V \times C \times V$ is a set of (*colored*) *edges*. Intuitively, vertices in V_1 are controlled by \mathcal{P}_1 and vertices in V_2 are controlled by \mathcal{P}_2 . We assume arenas to be *non-blocking*: for all $v \in V$, there exists $(v, c, v') \in E$. For $v \in V$, a *play of \mathcal{A} from v* is an infinite sequence of edges $\pi = (v_0, c_1, v_1)(v_1, c_2, v_2) \dots \in E^\omega$ such that $v_0 = v$. A *history of \mathcal{A} from v* is a finite prefix in E^* of a play of \mathcal{A} from v . For convenience, we assume that there is a distinct *empty history* λ_v for every $v \in V$. If $\gamma = (v_0, c_1, v_1) \dots (v_{n-1}, c_n, v_n)$ is a non-empty history of \mathcal{A} , we define $\text{last}(\gamma) = v_n$. For an empty history λ_v , we define $\text{last}(\lambda_v) = v$. For $i \in \{1, 2\}$, we denote by $\text{Hists}_i(\mathcal{A})$ the set of histories γ of \mathcal{A} such that $\text{last}(\gamma) \in V_i$. An arena is *finite* if V and E are finite. An arena $\mathcal{A} = (V, V_1, V_2, E)$ is a *one-player arena of \mathcal{P}_1* (resp. \mathcal{P}_2) if $V_2 = \emptyset$ (resp. $V_1 = \emptyset$).

Strategies. Let $i \in \{1, 2\}$. A *strategy of \mathcal{P}_i on \mathcal{A}* is a function $\sigma_i: \text{Hists}_i(\mathcal{A}) \rightarrow E$ such that for all $\gamma \in \text{Hists}_i(\mathcal{A})$, the first component of $\sigma_i(\gamma)$ coincides with $\text{last}(\gamma)$. Given a strategy σ_i of \mathcal{P}_i , we say that a play $\pi = e_1 e_2 \dots$ is *consistent with σ_i* if for all finite prefixes $\gamma = e_1 \dots e_n$ of π such that $\text{last}(\gamma) \in V_i$, $\sigma_i(\gamma) = e_{n+1}$. A strategy σ_i is *memoryless* (also called *positional* in the literature) if its outputs only depend on the current vertex and not on the whole history, i.e., if there exists a function $f: V_i \rightarrow E$ such that for all $\gamma \in \text{Hists}_i(\mathcal{A})$, $\sigma_i(\gamma) = f(\text{last}(\gamma))$.

Memory skeletons. A (*memory*) *skeleton* is a tuple $\mathcal{M} = (M, m_{\text{init}}, \alpha_{\text{upd}})$ such that M is a finite set of *states*, $m_{\text{init}} \in M$ is an *initial state*, and $\alpha_{\text{upd}}: M \times C \rightarrow M$ is an *update function*. Such skeletons are sometimes called *chromatic*, as their transitions are only based on the colors seen. We denote by α_{upd}^* the natural extension of α_{upd} to finite sequences of colors. We define the trivial skeleton $\mathcal{M}_{\text{triv}}$ as the only skeleton with a single state.

Let $\mathcal{M}_1 = (M_1, m_{\text{init}}^1, \alpha_{\text{upd}}^1)$ and $\mathcal{M}_2 = (M_2, m_{\text{init}}^2, \alpha_{\text{upd}}^2)$ be two skeletons. Their (*direct*) *product* $\mathcal{M}_1 \otimes \mathcal{M}_2$ is the skeleton $(M, m_{\text{init}}, \alpha_{\text{upd}})$ where $M = M_1 \times M_2$, $m_{\text{init}} = (m_{\text{init}}^1, m_{\text{init}}^2)$, and for all $m_1 \in M_1$, $m_2 \in M_2$, $c \in C$, $\alpha_{\text{upd}}((m_1, m_2), c) = (\alpha_{\text{upd}}^1(m_1, c), \alpha_{\text{upd}}^2(m_2, c))$.

For $\mathcal{M} = (M, m_{\text{init}}, \alpha_{\text{upd}})$ a skeleton, a strategy σ_i of \mathcal{P}_i is *based on \mathcal{M}* if there exists a function $\alpha_{\text{nxt}}: V \times M \rightarrow E$ such that for all vertices $v \in V_i$, $\sigma_i(\lambda_v) = \alpha_{\text{nxt}}(v, m_{\text{init}})$, and for all non-empty histories $\gamma = (v_0, c_1, v_1) \dots (v_{n-1}, c_n, v_n) \in \text{Hists}_i(\mathcal{A})$, $\sigma_i(\gamma) = \alpha_{\text{nxt}}(\text{last}(\gamma), \alpha_{\text{upd}}^*(m_{\text{init}}, c_1 \dots c_n))$. Such a strategy is said to use *chromatic memory*. Notice that a strategy is memoryless if and only if it is based on $\mathcal{M}_{\text{triv}}$.

Objectives. An *objective* is a set $W \subseteq C^\omega$ of infinite words. When an objective W is clear in the context, we say that an infinite word $w \in C^\omega$ is *winning* if $w \in W$, and *losing* if $w \notin W$. We write \overline{W} for the complement $C^\omega \setminus W$ of an objective W . An objective W is *prefix-independent* if for all $w \in C^*$ and $w' \in C^\omega$, $w' \in W$ if and only if $ww' \in W$. For a finite word $w \in C^*$, we write $w^{-1}W = \{w' \in C^\omega \mid ww' \in W\}$ for the *winning continuations* of w . We have in general that $\varepsilon^{-1}W = W$, and if W is prefix-independent, for all $w \in C^*$, we have $w^{-1}W = W$.

A *game* is a tuple (\mathcal{A}, W) , where \mathcal{A} is an arena and W is an objective. In such a game, \mathcal{P}_1 wants to achieve an infinite word in W through the infinite interaction with \mathcal{P}_2 in \mathcal{A} , while \mathcal{P}_2 wants to achieve an infinite word in \overline{W} .

ω -regular objectives. A central class of objectives is the one of *ω -regular objectives*. They admit multiple equivalent definitions: they are the objectives that can be expressed using an *ω -regular expression*, a *non-deterministic Büchi automaton*, a *deterministic parity automaton*... In this contribution, we mostly use their representation as a deterministic parity automaton (DPA). A (transition-based) DPA is a tuple $\mathcal{D} = (Q, q_{\text{init}}, \delta, p)$ where Q is a finite set of *states*, $q_{\text{init}} \in Q$ is an *initial state*, $\delta: Q \times C \rightarrow Q$ is a *transition function*, and $p: Q \times C \rightarrow \mathbb{N}$ is a *priority function*. A DPA \mathcal{D} recognizes an objective containing the words such that, when read in \mathcal{D} , the maximal priority that they see infinitely often is even. Notice that the first three components of a DPA are syntactically the same as a memory skeleton; for a memory skeleton $\mathcal{M} = (M, m_{\text{init}}, \alpha_{\text{upd}})$, if there is $p: M \times C \rightarrow \mathbb{N}$ such that $\mathcal{D} = (\mathcal{M}, p)$, we say that \mathcal{D} is *built on top of* \mathcal{M} .

Optimality. Let $\mathcal{A} = (V, V_1, V_2, E)$ be an arena, (\mathcal{A}, W) be a game, and $v \in V$. We say that a *strategy* σ_1 of \mathcal{P}_1 is *winning from* v if for all plays $(v_0, c_1, v_1)(v_1, c_2, v_2) \dots$ from v consistent with σ_1 , $c_1 c_2 \dots \in W$. A strategy of \mathcal{P}_1 is *optimal for* \mathcal{P}_1 *in* (\mathcal{A}, W) if it is winning from all the vertices from which \mathcal{P}_1 has a winning strategy. We often write *optimal for* \mathcal{P}_1 *in* \mathcal{A} if the objective W is clear from the context. This notion of optimality requires a *single* strategy to be winning from *all* the winning vertices (a property sometimes called *uniformity*).

Our games are *zero-sum*, hence the objective of \mathcal{P}_2 is formally \overline{W} . For the sake of readability, we still talk about winning and optimal strategies of \mathcal{P}_2 for W , taking into account the symmetric nature of their antagonistic role (i.e., an optimal strategy of \mathcal{P}_2 for W is an optimal strategy of \mathcal{P}_1 for \overline{W} if the two players are swapped).

Let W be an objective and $i \in \{1, 2\}$. A memory skeleton \mathcal{M} *suffices for* \mathcal{P}_i *for* W (resp. in finite, one-player arenas) if \mathcal{P}_i has an optimal strategy based on \mathcal{M} in game (\mathcal{A}, W) for all (resp. finite, one-player) arenas \mathcal{A} . We say that W is *\mathcal{M} -determined* (resp. in finite arenas) if \mathcal{M} suffices for both \mathcal{P}_1 and \mathcal{P}_2 for W (resp. in finite arenas), and that W is *finite-memory-determined* if it is \mathcal{M} -determined for some \mathcal{M} . We call $\mathcal{M}_{\text{triv}}$ -determinacy *memoryless determinacy*. For consistency with the literature [30], we call the notion “ $\mathcal{M}_{\text{triv}}$ suffices to play optimally for \mathcal{P}_1 for W ” *half-positionality of* W .

► **Remark 1.** We stress that the notion of finite-memory determinacy used throughout the paper is strong in several respects: it requires *chromatic memory* to be sufficient (the memory can only observe colors, and not actual edges that are taken during a play), and it requires the *same* memory skeleton to be sufficient in all arenas (that is, it is *arena-independent*). ◻

3 Characterization of finite-memory-determined objectives

Many objectives are known to be memoryless-determined, that is, to require no memory except the knowledge of the current vertex to be won. For instance, Ehrenfeucht and Mycielski proved in 1979 that mean-payoff games are memoryless-determined in finite arenas [19], and Emerson and Jutla proved in 1991 that parity games are memoryless-determined [21].

It has therefore been a natural research direction to try to characterize the winning objectives that are memoryless-determined. Gimbert and Zielonka gave the first characterization of winning objectives that are memoryless-determined in finite arenas [24]; this characterization is presented in Section 3.1.1. Trying to extend that result to finite memory was very natural, but the extension could only be handled fifteen years later, and required to focus on chromatic arena-independent memory [4]; this is presented in Section 3.1.2.

Some objectives are memoryless-determined in finite arenas, but require infinite memory in infinite arenas; this is for instance the case of mean-payoff objectives. Hence the above characterizations do not carry over to infinite arenas. Inspired by a work by Colcombet and Niwiński [16] who focused on prefix-independent and memoryless-determined objectives, a full characterization of objectives that are finite-memory-determined in infinite arenas is given in [7], where it is proved that they actually coincide with ω -regular objectives; this is discussed in Section 3.2.

3.1 Finite graph games

3.1.1 Memoryless-determined objectives

The first complete characterization of objectives (or more generally *preference relations* – we do not define this notion here) admitting memoryless optimal strategies is established in [24]. By complete characterization, we mean sufficient and necessary conditions on the objectives. This result can be stated as follows.

► **Theorem 2** ([24, Theorem 2]). *An objective W is memoryless-determined if and only if both W and \bar{W} are monotone and selective.*

Roughly, monotony for an objective says that if an infinite word uw_1^ω is winning and another one uw_2^ω is losing, then their “winning status” cannot be swapped by replacing prefix u , i.e., we cannot have $u'w_1^\omega$ losing and $u'w_2^\omega$ winning for any $u' \in C^*$. This property is obviously satisfied by prefix-independent objectives, but is more general.

Selectivity is defined with regard to cycle mixing: starting from two sequences of colors, it is impossible to create a third one by mixing the first two in such a way that the third one is winning while the first two are not. Similar-looking notions (*fairly mixing*, *concave* [23, 30]) had been defined in other attempts in the literature, but they slightly differ and are actually incomparable to selectivity.

A by-product of the proof of the above theorem is the following so-called *one-to-two-player lift*.

► **Corollary 3.** *Assume that for an objective W , memoryless strategies suffice for both \mathcal{P}_1 and \mathcal{P}_2 in their respective finite one-player arenas. Then W is memoryless-determined in finite arenas.*

Such a lifting corollary provides a neat and easy way to prove that an objective admits memoryless optimal strategies without proving monotony and selectivity at all: proving it in the two one-player subcases, which is generally much easier as it boils down to graph reasoning, and then lifting the result to the general two-player case through the corollary.

► **Example 4.** Let $C = \mathbb{Q}$. For $w = c_1c_2\dots \in C^\omega$, we define its *mean payoff*

$$\text{MP}(w) = \liminf_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n c_i$$

as the limit (inferior) of the average of the colors of its finite prefixes. We define the objective “achieving a non-negative mean payoff” as $\text{MP}_{\geq 0} = \{w \in C^\omega \mid \text{MP}(w) \geq 0\}$. This objective is memoryless-determined in finite arenas, which was first proved in [19]. We argue informally that it is easy to recover this result using Corollary 3.

We take the point of view of \mathcal{P}_1 . Let \mathcal{A} be a finite *one-player* arena of \mathcal{P}_1 . We consider the cycles of \mathcal{A} , and we say that a cycle is *non-negative* if the average of its colors is non-negative. Clearly, \mathcal{P}_1 can win for $\text{MP}_{\geq 0}$ if a non-negative cycle is reachable: \mathcal{P}_1 can simply reach the cycle and loop around it. To win with a memoryless strategy, we simply observe that a non-negative cycle always contains a non-negative *simple* cycle (i.e., not going twice through the same vertex). A *simple* cycle can be reached and looped around with a *memoryless* strategy. This describes a way to win with a memoryless strategy if there is a non-negative cycle, and this is actually “complete”: if there is no non-negative cycle, then \mathcal{P}_1 simply cannot win for $\text{MP}_{\geq 0}$. We have shown that given a fixed initial vertex of \mathcal{A} , if \mathcal{P}_1 can win, then \mathcal{P}_1 can win with a memoryless strategy. Formally, we must still argue that we can build a single memoryless strategy winning “uniformly” from all the vertices from which \mathcal{P}_1 has a winning strategy, which we do not do here but is reasonably straightforward. These arguments show that memoryless strategies suffice for \mathcal{P}_1 in its one-player arenas, and the same reasoning works for \mathcal{P}_2 (using negative cycles instead). By Corollary 3, $\text{MP}_{\geq 0}$ is memoryless-determined in finite arenas. ┘

3.1.2 Finite-memory-determined objectives

Gimbert and Zielonka’s result completely characterizes objectives which can be won with memoryless strategies in all games played on finite arenas. This paves the way to the quest for a similar characterization for finite-memory strategies. A first reasonable attempt for a generalization to finite-memory strategies could be: given an objective, if in all finite one-player arenas, players have finite-memory optimal strategies, does the same hold in finite two-player arenas? Unfortunately, this generalization does not actually hold: there are objectives for which both players have finite-memory optimal strategies in their respective finite one-player arenas, but for which there exists a finite two-player arena that requires infinite memory for a player to win – see [4, Figure 1].

However, a result similar to Theorem 2 can be proved for \mathcal{M} -*determinacy*. Note the subtlety here: \mathcal{M} -determinacy requires the memory skeleton \mathcal{M} to be uniform with regard to the arena; the structure of the memory must be *arena-independent*, as opposed to the more general version sketched above where the memory may depend on the arena.

► **Theorem 5** ([4, Theorem 3.6]). *Let \mathcal{M} be a memory skeleton. An objective W is \mathcal{M} -determined if and only if both W and its complement \overline{W} are \mathcal{M} -monotone and \mathcal{M} -selective.*

The two concepts of \mathcal{M} -monotony and \mathcal{M} -selectivity are keys to the approach. Intuitively, they correspond to Gimbert and Zielonka’s monotony and selectivity, modulo a memory skeleton. The more general concepts of \mathcal{M} -monotony and \mathcal{M} -selectivity serve the same purpose, but they only compare sequences of colors that are deemed equivalent by the memory skeleton. For the sake of illustration, take selectivity: it implies that one has no interest in mixing different cycles of the game arena. For its generalization, the memory

skeleton is taken into account: \mathcal{M} -selectivity implies that one has no interest in mixing cycles of the game arena that are read as cycles on the same memory state in the skeleton \mathcal{M} . In particular, $\mathcal{M}_{\text{triv}}$ -monotony and $\mathcal{M}_{\text{triv}}$ -selectivity are respectively equivalent to the original notions of monotony and selectivity, as $\mathcal{M}_{\text{triv}}$ never distinguishes sequences of colors.

The proof of this theorem mimics the one of [24] via the notion of \mathcal{M} -covered arena. An \mathcal{M} -covered arena is an arena which is compatible with \mathcal{M} , in the sense that there is a morphism from the arena to \mathcal{M} . Examples of \mathcal{M} -covered arenas are products of an arena with \mathcal{M} , but they are more general (notably, removing edges from these products preserves \mathcal{M} -coverability). Strategies based on \mathcal{M} on arenas correspond to memoryless strategies on \mathcal{M} -covered arenas. The proof technique for memoryless strategies, relying on an induction on the size of arenas, can be made on \mathcal{M} -covered arenas to obtain the theorem for strategies based on \mathcal{M} .

As for memoryless strategies, a by-product of the proof of the above theorem is a one-to-two-player lift.

► **Corollary 6.** *Let \mathcal{M} be a memory skeleton. Assume that for an objective W , strategies based on \mathcal{M} suffice for both \mathcal{P}_1 and \mathcal{P}_2 in their respective finite one-player arenas. Then W is \mathcal{M} -determined in finite arenas.*

As in general \mathcal{P}_1 and \mathcal{P}_2 may need different memory structures, we may instantiate the result with two different memory structures \mathcal{M}_1 and \mathcal{M}_2 : if \mathcal{M}_1 suffices for \mathcal{P}_1 and \mathcal{M}_2 suffices for \mathcal{P}_2 for W in finite one-player arenas, then $\mathcal{M}_1 \otimes \mathcal{M}_2$ suffices for both \mathcal{P}_1 and \mathcal{P}_2 in their finite one-player arenas (remembering more information cannot do harm), so W is $(\mathcal{M}_1 \otimes \mathcal{M}_2)$ -determined in finite arenas. However, it is unknown whether \mathcal{M}_1 (resp. \mathcal{M}_2) alone could be sufficient for \mathcal{P}_1 (resp. \mathcal{P}_2) in finite (two-player) arenas, or if the product is required.



■ **Figure 1** Memory skeletons \mathcal{M} (left) and \mathcal{M}' (right) for two-target reachability games. In figures, diamonds (resp. circles, squares) represent memory or automaton states (resp. arena vertices controlled by \mathcal{P}_1 , arena vertices controlled by \mathcal{P}_2).

► **Example 7.** Consider the objective $W = (C^*aC^\omega) \cap (C^*bC^\omega)$ over alphabet $C = \{a, b, c\}$. The objective W requires that both colors a and b should be seen at least once. Consider the two skeletons \mathcal{M} and \mathcal{M}' in Figure 1.

Let us briefly explain why W is not $\mathcal{M}_{\text{triv}}$ -monotone (i.e., monotone) but is \mathcal{M} -monotone. On the one hand, $ab^\omega \in W$ is preferred to $aa^\omega \notin W$, but $ba^\omega \in W$ is preferred to $bb^\omega \notin W$; hence, W is not $\mathcal{M}_{\text{triv}}$ -monotone. On the other hand, skeleton \mathcal{M} distinguishes a and b (in the sense that they reach two different states of skeleton \mathcal{M}), hence we do not need to compare their winning continuations $a^{-1}W$ and $b^{-1}W$. Also, we can prove that two pairs of continuations $u_1^{-1}W$ and $u_2^{-1}W$ such that u_1 and u_2 reach the same memory state of \mathcal{M} are comparable (for the inclusion), hence W is \mathcal{M} -monotone.

Let us now briefly explain why W is not $\mathcal{M}_{\text{triv}}$ -selective (i.e., selective) but is \mathcal{M}' -selective. Notice that a and b are *losing cycles*, in the sense that if repeated infinitely often, they generate a losing word ($a^\omega, b^\omega \in \overline{W}$). But combining a and b , which are cycles on the state of $\mathcal{M}_{\text{triv}}$ (as are all finite words), may result in a winning word. For instance, they can be used

to make $(ab)^\omega \in W$. Hence, W is not $\mathcal{M}_{\text{triv}}$ -selective. On the other hand, W is \mathcal{M}' -selective, as \mathcal{M}' distinguishes cycles in such a way that two losing cycles on the same memory state cannot be combined into a winning cycle. For m'_1 : all cycles around m'_1 are losing (since they contain neither a nor b) and cannot be combined into a winning cycle. For m'_2 : if one reaches m'_2 , it means that one of a or b was seen, hence only one color remains to be seen. Any subsequent cycle on m'_2 is either useless (if it sees c or the color among a and b that was already seen) or immediately winning (if it sees the color among a and b that was not seen). There is therefore no advantage to combine multiple cycles once m'_2 is reached.

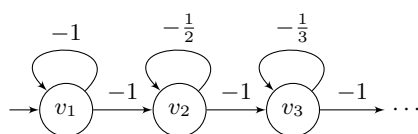
Overall, W is \mathcal{M} -monotone and \mathcal{M}' -selective, and we can show in a similar fashion that \overline{W} is \mathcal{M} -monotone and $\mathcal{M}_{\text{triv}}$ -selective. Moreover, monotony and selectivity are “preserved by product” (intuitively, as having more information to our disposal is never harmful), so both W and \overline{W} are $(\mathcal{M} \otimes \mathcal{M}')$ -monotone and $(\mathcal{M} \otimes \mathcal{M}')$ -selective. This allows to conclude by Theorem 5 that W is $(\mathcal{M} \otimes \mathcal{M}')$ -determined in finite arenas. Skeleton $\mathcal{M} \otimes \mathcal{M}'$ has formally four states but only three reachable states, so both players can play optimally for W using three memory states, which is minimal [22]. \square

Applicability of these results goes beyond ω -regular objectives. For instance, the intersection of a reachability condition and a mean-payoff objective requires two memory states, and it suffices for both players to keep track of whether the reachability objective has already been satisfied, e.g., using skeleton \mathcal{M} of Figure 1.

3.2 Infinite graph games

Unfortunately, memoryless determinacy in finite arenas does not carry over straightforwardly to infinite arenas. Indeed, there are objectives that are memoryless-determined in finite arenas but that require infinite memory in some infinite (even one-player) arenas: this is the case for the mean-payoff objective $\text{MP}_{\geq 0}$, which we showed to be memoryless-determined in finite arenas in Example 4.

► **Example 8.** We consider objective $\text{MP}_{\geq 0}$ and the infinite one-player arena of \mathcal{P}_1 in Figure 2 [40]. Despite the fact that all colors are negative, \mathcal{P}_1 has a winning strategy: the idea is to loop on state s_i sufficiently many times to bring the payoff close to $-\frac{1}{i}$, and then move to s_{i+1} and repeat. At the limit, the mean payoff is 0. However, this winning strategy requires memory (even *infinite* memory) to be implemented, and no memoryless strategy is winning. Hence, $\text{MP}_{\geq 0}$ is not memoryless-determined in infinite arenas. \square



■ **Figure 2** One-player arena requiring infinite memory for objective $\text{MP}_{\geq 0}$.

An interesting result dealing with infinite arenas is the work by Colcombet and Niwiński [16], who proved that a prefix-independent objective W is memoryless-determined in infinite arenas if and only if W is a *parity condition*. A parity condition is an objective recognized by a deterministic parity automaton with a single state, in which each color is directly mapped to a priority in a finite set of integers. In [7], we generalized it to a characterization of objectives that are finite-memory-determined in infinite arenas. Formulating this generalization requires the language-theoretic notion of *right congruence*. Let $W \subseteq C^\omega$

be an objective. The *right congruence* $\sim_W \subseteq C^* \times C^*$ of W is defined as $w_1 \sim_W w_2$ if $w_1^{-1}W = w_2^{-1}W$ (meaning that w_1 and w_2 have the same winning continuations). When \sim_W has finitely many equivalence classes, we can associate to W a natural skeleton \mathcal{M}_W whose set of states is the set of equivalence classes and with transitions defined in a natural way [43, 35]. We call skeleton \mathcal{M}_W the *prefix classifier of W* , as two finite words reach the same state of \mathcal{M}_W if and only if they are equivalent for \sim_W . We can now state the main result from [7].

► **Theorem 9** ([7]). *An objective is finite-memory-determined (in arenas of any cardinality) if and only if it is ω -regular. Furthermore, if W is \mathcal{M} -determined (in arenas of any cardinality), then W is recognized by a DPA built on top of $\mathcal{M} \otimes \mathcal{M}_W$.*

Before [7], ω -regular objectives were known to be finite-memory-determined [21, 45]: if it is possible to represent an objective with a DPA, then the structure of this automaton suffices to play optimally for both players. Indeed, keeping in memory the extra information from this DPA effectively reduces any game using this objective into a (larger) game using a (simpler) parity condition. This shows one implication of Theorem 9. The proof of the other implication goes through the following steps: if W is \mathcal{M} -determined for some skeleton \mathcal{M} , then

- W is \mathcal{M} -cycle-consistent: after any finite word, if we concatenate infinitely many winning (resp. losing) cycles on the skeleton state reached by that word, then it only produces winning (resp. losing) infinite words;
- \mathcal{M}_W is finite, which implies that W is \mathcal{M}_W -prefix-independent: \mathcal{M}_W classifies prefixes in such a way that two prefixes reaching the same memory state have the same winning continuations.

From this, we obtain in particular that W is $(\mathcal{M} \otimes \mathcal{M}_W)$ -cycle-consistent and $(\mathcal{M} \otimes \mathcal{M}_W)$ -prefix-independent. We can prove that under these properties, one can associate to transitions of $\mathcal{M} \otimes \mathcal{M}_W$ priorities such that W is recognized by a DPA built on top of $\mathcal{M} \otimes \mathcal{M}_W$. This part of the proof is rather technical, but relies on ordering the cycles according to “how good they are for winning”; order which can be used to assign priorities to transitions.

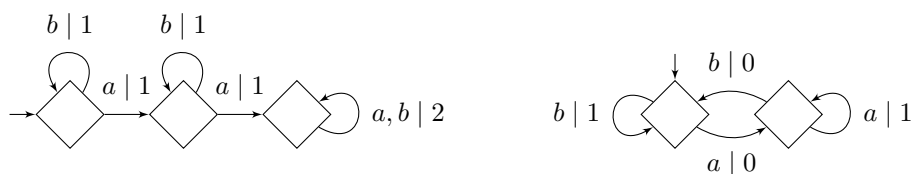
We recover the result of [16], since the prefix classifier has a single state in the case of a prefix-independent objective (that is, the prefix classifier is $\mathcal{M}_{\text{triv}}$). Hence, under the assumption that W is prefix-independent and memoryless-determined (that is, $\mathcal{M}_{\text{triv}}$ -determined), we deduce that W is recognized by a DPA built on top of the skeleton $\mathcal{M}_{\text{triv}} \otimes \mathcal{M}_{\text{triv}}$ with a single state, that is, W is a parity condition.

As previously, we can extract a one-to-two-player lift from the proof of the above result.

► **Corollary 10.** *Let W be an objective. If \mathcal{M} suffices for both \mathcal{P}_1 and \mathcal{P}_2 in their respective one-player arenas (of arbitrary cardinality), then W is $(\mathcal{M} \otimes \mathcal{M}_W)$ -determined (in arenas of arbitrary cardinality). In particular, if W is prefix-independent, then under the previous hypotheses, W is \mathcal{M} -determined.*

We discuss two ω -regular objectives and illustrate that to represent them using DPAs, we may need some information given by their prefix classifier and some information given by a sufficient memory structure.

► **Example 11.** Consider the objective $W_1 = b^*ab^*aC^\omega$ over the alphabet $C = \{a, b\}$. This objective has a prefix classifier \mathcal{M}_{W_1} with three states, corresponding to the three equivalence classes of finite words that have seen 0, 1, or 2 times the color a . This objective is also memoryless-determined that is, $\mathcal{M}_{\text{triv}}$ -determined; we do not prove it here (in finite arenas,



■ **Figure 3** DPA recognizing $W_1 = b^*ab^*aC^\omega$ (left), which is built on top of its prefix classifier \mathcal{M}_{W_1} . DPA recognizing $W_2 = C^*(ab)^\omega$ (right), which is built on top of a minimal memory structure sufficient for \mathcal{P}_1 and \mathcal{P}_2 . A transition from a state q to a state q' labeled with “ $c | n$ ” means that $\delta(q, c) = q'$ and that $p(q, c) = n$.

it can be shown with Corollary 3). By Theorem 9, this means that W_1 can be recognized by a DPA built on top $\mathcal{M}_{\text{triv}} \otimes \mathcal{M}_{W_1} = \mathcal{M}_{W_1}$. We represent such a DPA in Figure 3 (left): its structure is the same as \mathcal{M}_{W_1} , and we just added the right priorities to its transitions.

Consider the objective $W_2 = C^*(ab)^\omega$ over the alphabet $C = \{a, b\}$. It is prefix-independent, hence its prefix classifier is $\mathcal{M}_{\text{triv}}$. It is furthermore \mathcal{M} -determined, where \mathcal{M} is the skeleton with two states that remembers whether a or b was last seen; this skeleton has the same structure as the DPA in Figure 3 (right). By Theorem 9, W_2 can be recognized by a DPA built on top of $\mathcal{M} \otimes \mathcal{M}_{\text{triv}} = \mathcal{M}$. We represent such a DPA in Figure 3 (right). ◻

On the other hand, our results can also illustrate why an objective is not finite-memory-determined (or equivalently, ω -regular).

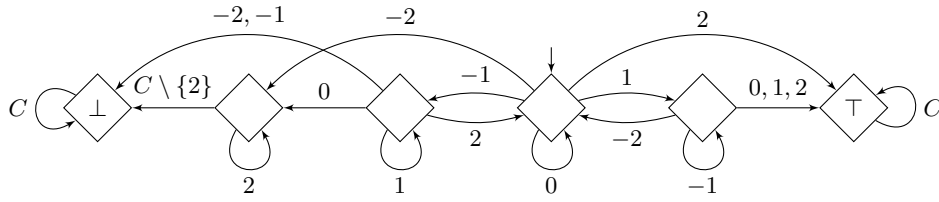
► **Example 12.** We go back to the mean-payoff objective $\text{MP}_{\geq 0}$ defined in Example 4. It is a prefix-independent objective, hence $\mathcal{M}_{\text{MP}_{\geq 0}} = \mathcal{M}_{\text{triv}}$. However, it is not $\mathcal{M}_{\text{triv}}$ -cycle-consistent. Indeed, repeating ad infinitum the same color from the set $\{-\frac{1}{n} \mid n \in \mathbb{N}_{>0}\}$ results in a losing word; however, the sequence $(-1)(-\frac{1}{2})(-\frac{1}{3}) \dots$ combining multiple such losing colors is winning (its mean payoff is exactly 0). This argument can be extended to any *finite* memory skeleton \mathcal{M} to show that $\text{MP}_{\geq 0}$ is not \mathcal{M} -cycle-consistent. This means that $\text{MP}_{\geq 0}$ is not finite-memory-determined over infinite arenas, and is not ω -regular. ◻

In some classes of objectives, finite-memory determinacy/ ω -regularity depends on the values of some parameters.

► **Example 13.** Let $C \subseteq \mathbb{Q}$ be a bounded set of colors and $\lambda \in (0, 1)$ be a *discount factor*. We consider the *discounted-sum objective* [42]

$$\text{DS}_{\geq 0}^{C, \lambda} = \{c_1c_2 \dots \in C^\omega \mid \sum_{i=1}^{\infty} c_i \cdot \lambda^{i-1} \geq 0\}.$$

For all values of λ , it is possible to show that $\text{DS}_{\geq 0}^{C, \lambda}$ is $\mathcal{M}_{\text{triv}}$ -cycle-consistent. This means by Theorem 9 that the prefix classifier of $\text{DS}_{\geq 0}^{C, \lambda}$, when *finite*, can be used as a DPA to recognize objective $\text{DS}_{\geq 0}^{C, \lambda}$. Proof details and a characterization of the values of C and λ such that $\text{DS}_{\geq 0}^{C, \lambda}$ is ω -regular can be found in [7, Section 4.1]. We give a specific example of values that make $\text{DS}_{\geq 0}^{C, \lambda}$ ω -regular: for $\lambda = \frac{1}{2}$ and $C = \{-2, -1, 0, 1, 2\}$, the prefix-classifier is finite and is depicted in Figure 4. ◻



■ **Figure 4** Prefix classifier of $DS_{\geq 0}^{C,\lambda}$ for $\lambda = \frac{1}{2}$ and $C = \{-2, -1, 0, 1, 2\}$. An infinite word is winning if and only if it does not reach state \perp .

4 Memory requirements of ω -regular objectives

Section 3.2 presented an equivalence between ω -regularity and a kind of finite-memory determinacy of two-player zero-sum games. This suggests that, in addition to their relevance in logic and in synthesis, understanding the *memory requirements of ω -regular objectives* is a natural stepping stone in order to study the strategy complexity of all two-player zero-sum games. We discuss known results about memory requirements of ω -regular objectives in this section.

As was stated in Theorem 9, one can relate the memory requirements of an ω -regular objective to its representation as a DPA. We may wonder how close this result brings us to characterizing precisely the memory requirements of ω -regular objectives. Albeit being quite general, there are still multiple questions about memory requirements that Theorem 9 is not able to answer precisely. We introduce these questions by highlighting two of its limitations.

The first limitation comes from the asymmetry of its two implications. In one direction, which is the novel contribution from Theorem 9, we start from a sufficient memory skeleton \mathcal{M} for some objective and show that the objective can then be represented as a DPA built on the automatic structure $\mathcal{M} \otimes \mathcal{M}_W$. In the other direction, we simply use the known memoryless-determinacy of parity conditions, which we already mentioned. What does this tell us on the memory requirements of an ω -regular objective? We know two things: (i) a minimal memory skeleton has always at most as many states as any DPA representing the objective; (ii) a minimal memory skeleton and a minimal DPA differ at most by the factor \mathcal{M}_W . However, we do not know in general how to get minimal memory requirements from a representation as a DPA.

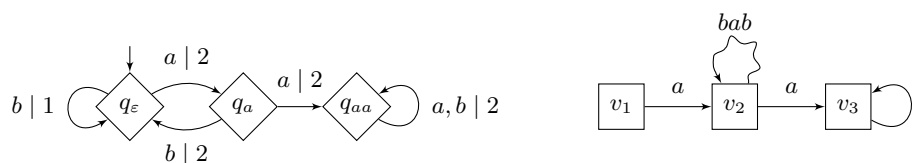
The second, perhaps more fundamental limitation is that it makes an assumption on the memory requirements of *both* players simultaneously, as it asks for a memory skeleton sufficient for both players. This assumption therefore conceals a possibly large gap between the individual memory requirements of the two players. This limitation also applies to Theorem 5 (which deals with games played on finite arenas), which cannot be used in general to give tight bounds about memory requirements of each individual player in two-player games.

We illustrate these two limitations on a small example: in this example, the representation of an objective as a DPA is an upper (but not a tight) bound on the memory requirements, and the memory requirements of each player differ.

► **Example 14.** Let $C = \{a, b\}$. We consider the objective $W = (b^*a)^\omega \cup (C^*aaC^\omega)$ of words that see a infinitely often or see a twice in a row at some point. This objective is recognized by the DPA with three states depicted in Figure 5 (left), and it is not possible to recognize it using a DPA with fewer states. We therefore know that both players can play optimally

using three states of memory, using the memory skeleton underlying this automaton. The prefix classifier of this objective has three states corresponding to three classes of finite words, and its structure also corresponds to the underlying structure of the DPA in Figure 5. According to Theorem 9, this suggests that \mathcal{P}_1 and \mathcal{P}_2 may need between one and three states of memory to play optimally.

It turns out here that \mathcal{P}_1 can play optimally with just one state of memory in all arenas (i.e., W is half-positional), which can be proved using results from [3] (discussed below). Meanwhile, \mathcal{P}_2 cannot play optimally with just one state of memory, as is witnessed by the arena in Figure 5 (right). If the play starts in v_1 , we observe that \mathcal{P}_2 loses by not using the loop in v_2 and going immediately to v_3 , as well as by staying infinitely often in v_2 . Player \mathcal{P}_2 can actually win, but needs to loop at least once (and finitely many times) in v_2 before going to v_3 , which cannot be done without memory. For this objective, \mathcal{P}_2 can actually play optimally with two states of memory: intuitively, \mathcal{P}_2 must keep track of whether the current history is in q_ε or q_a , but there is no point in keeping track of state q_{aa} , as the play is already lost in that state for \mathcal{P}_2 . \lrcorner



■ **Figure 5** A DPA representing the objective $W = (b^*a)^\omega \cup (C^*aaC^\omega)$ from Example 14 (left), and an arena in which \mathcal{P}_2 cannot play optimally with a memoryless strategy (right).

The following questions therefore remain: given an ω -regular objective, what is a *minimal* (i.e., with as few states as possible) memory skeleton sufficient to play optimally for both players? And for a *single* player? A less ambitious (but still open) question would be to understand the memoryless case: how to characterize/decide memoryless determinacy or half-positionality? And would these precise results give us even more information on the representation of ω -regular objectives, perhaps using other acceptance conditions than the parity one? Progress toward these questions, which can be seen as strengthenings of Theorem 9, has been obtained on specific classes of ω -regular objectives. We discuss two of them here.

Memory requirements of Muller conditions. Muller conditions are objectives whose winning words depend solely on the set of colors seen *infinitely often*. They are usually specified by a set $\mathcal{F} \subseteq 2^C$ of sets of colors. The related Muller condition then contains the set of words $w \in C^\omega$ such that the set of colors seen infinitely often by w is a set of \mathcal{F} . They are in particular prefix-independent, i.e., their prefix classifier has just one state. A systematic study of the memory requirements of Muller conditions started in the '80s, with first general upper bounds through the later appearance record construction [26, 37], culminating in a complete characterization of their (*chaotic*) memory requirements [18].

More relevant to our chromatic memory considerations, we mention the recent work by Casares [11] that characterizes the chromatic memory requirements of Muller conditions for each individual player. The characterization uses the *Rabin* acceptance condition, which subsumes parity acceptance conditions.

► **Theorem 15** ([11, Theorem 27]). *Let W be a Muller condition and \mathcal{M} be a memory skeleton. Structure \mathcal{M} suffices for \mathcal{P}_1 for objective W if and only if W is recognized by a deterministic Rabin automaton built on top of \mathcal{M} .*

Once again, one direction has been known for some time: Rabin conditions have been known to be half-positional for some time [29] (but unlike parity conditions, their complement may not be). The other, novel direction was obtained thanks to results about the representation of ω -regular objectives [12].

This provides, in the special case of Muller conditions, a characterization of the memory requirements of each player without any blow-up in any direction of the equivalence, and independently of the memory requirements of the other player. It therefore goes beyond the two limitations of Theorem 9 sketched above. As a bonus, this result allows to link the problem of finding a minimal memory skeleton to the problem of minimizing a Rabin automaton, and implies that the related decision problem (given a Muller condition, is there a sufficient memory skeleton with $\leq k$ states for a fixed k ?) is NP-complete.

Half-positional deterministic Büchi automata. *Deterministic Büchi automata* (DBAs), unlike their nondeterministic counterparts, only recognize a proper subclass of the ω -regular objectives [44]. They can be seen as a special case of DPAs using only priorities 1 and 2 (the automaton in Figure 5 is also a DBA). The objectives that they recognize are incomparable to Muller conditions. In particular, they recognize some non-prefix-independent objectives, hence with a non-trivial prefix classifier. Currently, their complete memory requirements are not understood – only their half-positionality has been fully characterized. Article [3] gives a characterization of the objectives that are half-positional among those that can be recognized by a DBA. This characterization is a conjunction of three properties that are decidable in polynomial time. The first property is equivalent for this class of objectives to the aforementioned *monotony* property, so we do not discuss it here.

The second property deals with the notion of *progress*: a finite word w_2 is said to be a progress after a finite word w_1 if w_1w_2 has strictly more winning continuations than w_1 . We illustrate this property on the objective from Example 14. We take the point of view of \mathcal{P}_2 , who wants to avoid seeing infinitely many a and avoid seeing a twice in a row at some point. If $w_1 = a$ and $w_2 = bab$, for \mathcal{P}_2 , w_2 is a progress after w_1 : any winning continuation of w_1 is still winning after w_1w_2 , and $w_1w_2ab^\omega$ is winning, while w_1ab^ω is losing. See the link between our choice of words and the arena in Figure 5. The reason that \mathcal{P}_2 needs memory here is that although w_2 is a progress after w_1 , repeating $w_1w_2^\omega$ is not winning. Hence, it is useful to play w_2 , but an optimal strategy cannot just repeat w_2 . We define a *progress-consistent* objective as an objective such that for all progresses w_2 after some w_1 , $w_1w_2^\omega$ is winning. This is necessary for half-positionality.

The third property relates once more to the representation of an ω -regular objective, this time using a DBA. Any deterministic automaton representing an ω -regular objective W needs at least one state per equivalence class of the right congruence \sim_W . The celebrated Myhill-Nerode theorem [38] states that to represent *regular* languages (of finite words), we do not need more than one state per equivalence class. This does not hold in general for ω -regular objectives, as was shown with objective $W_2 = C^*(ab)^\omega$ in Example 11. Still, *some* ω -regular objectives admit representations using exactly one state per equivalence class [1]. One example was given in Example 14, which admits a representation as a DBA with three states – one per equivalence class. One of the main technical contribution of [3] is that for an objective recognizable by a DBA to be half-positional, it is necessary that it admits a representation as a DBA with just one state per equivalence class. In such cases, a DBA can be minimized in polynomial time.

► **Theorem 16** ([3, Theorem 10]). *Let W be an objective recognized by a DBA. Objective W is half-positional if and only if it is monotone, progress-consistent, and recognized by a DBA built on top of its prefix classifier.*

5 Perspectives

We highlight some of the remaining open paths in the topic of strategy complexity of zero-sum games on graphs. As shown in Section 4, the memory requirements of ω -regular objectives, despite their relevance to logic and synthesis, are not fully mapped out. As discussed in Section 1, there are two relevant memory formalisms for this question: the first one is the chromatic one (which we discussed extensively), and the second one is the *chaotic* memory formalism. A promising research direction about *chaotic* memory requirements was recently given in [13], which proved a bridge between this question and the theory of *good-for-games* automata for Muller conditions. Moreover, it is shown that for some Muller conditions, chaotic memory requirements (expressed as a number of memory *states*) can be exponentially smaller than chromatic ones, at the cost of having to specialize the transitions of the memory structure for each distinct arena.

We also mention a recent tool used to study zero-sum games: *universal graphs*. Universal graphs were originally introduced to design algorithms to decide the winner for some classes of games [15]. Recently, Ohlmann [39] showed an equivalence between the existence of some “good” universal graph for an objective and half-positionality of the objective. Roughly, a universal graph (with the right properties) can be a structural witness of the half-positionality of an objective, and any half-positional objective has a good universal graph. This result has already been used to prove the sufficient condition of Theorem 16 [3] by mechanizing the construction of good universal graphs given a DBA with the right properties. This suggests that using universal graphs may be one path toward understanding half-positionality of ω -regular objectives.

References

- 1 Dana Angluin and Dana Fisman. Regular ω -languages with an informative right congruence. *Inf. Comput.*, 278:104598, 2021. doi:10.1016/j.ic.2020.104598.
- 2 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018. doi:10.1007/978-3-319-10575-8_27.
- 3 Patricia Bouyer, Antonio Casares, Mickael Randour, and Pierre Vandenhover. Half-positional objectives recognized by deterministic Büchi automata. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CONCUR.2022.20.
- 4 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhover. Games where you can play optimally with arena-independent finite memory. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:11)2022.
- 5 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Inf.*, 55(2):91–127, 2018. doi:10.1007/s00236-016-0274-1.
- 6 Patricia Bouyer, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhover. Arena-independent finite-memory determinacy in stochastic games. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.26.
- 7 Patricia Bouyer, Mickael Randour, and Pierre Vandenhover. Characterizing omega-regularity through finite-memory determinacy of games on infinite graphs. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.STACS.2022.16.

- 8 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications – 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016. doi:10.1007/978-3-319-30000-9_1.
- 9 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.
- 10 Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin. Energy mean-payoff games. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 21:1–21:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.21.
- 11 Antonio Casares. On the minimisation of transition-based Rabin automata and the chromatic memory requirements of Muller conditions. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CSL.2022.12.
- 12 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using Muller conditions. 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 123:1–123:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.123.
- 13 Antonio Casares, Thomas Colcombet, and Karoliina Lehtinen. On the size of good-for-games Rabin automata and its link with the memory in Muller games. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229, pages 117:1–117:20, 2022. doi:10.4230/LIPICs.ICALP.2022.117.
- 14 Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Inf.*, 51(3-4):129–163, 2014. doi:10.1007/s00236-013-0182-6.
- 15 Thomas Colcombet, Nathanaël Fijalkow, Pawel Gawrychowski, and Pierre Ohlmann. The theory of universal graphs for infinite duration games. *CoRR*, abs/2104.05262, 2021. arXiv:2104.05262.
- 16 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theor. Comput. Sci.*, 352(1-3):190–196, 2006. doi:10.1016/j.tcs.2005.10.046.
- 17 Florent Delgrange, Joost-Pieter Katoen, Tim Quatmann, and Mickael Randour. Simple strategies in multi-objective MDPs. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 346–364. Springer, 2020. doi:10.1007/978-3-030-45190-5_19.
- 18 Stefan Dziembowski, Marcin Jurdzinski, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 – July 2, 1997*, pages 99–110. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614939.
- 19 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979. doi:10.1007/BF01768705.

- 20 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. In *FOCS*, pages 328–337. IEEE Computer Society, 1988. doi:10.1109/SFCS.1988.21949.
- 21 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.
- 22 Nathanaël Fijalkow and Florian Horn. The surprising complexity of reachability games. *CoRR*, abs/1010.2420, 2010. doi:10.48550/arXiv.1010.2420.
- 23 Hugo Gimbert and Wiesław Zielonka. When can you play positionally? In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–697. Springer, 2004. doi:10.1007/978-3-540-28629-5_53.
- 24 Hugo Gimbert and Wiesław Zielonka. Games where you can play optimally without any memory. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 – Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi:10.1007/11539452_33.
- 25 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 26 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 60–65. ACM, 1982. doi:10.1145/800070.802177.
- 27 Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Good-for-MDPs automata for probabilistic analysis and reinforcement learning. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2020. doi:10.1007/978-3-030-45190-5_17.
- 28 Florian Horn. Random fruits on the Zielonka tree. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPICs*, pages 541–552. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009. doi:10.4230/LIPICs.STACS.2009.1848.
- 29 Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Ann. Pure Appl. Log.*, 69(2-3):243–268, 1994. doi:10.1016/0168-0072(94)90086-8.
- 30 Eryk Kopczyński. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006. doi:10.1007/11787006_29.
- 31 Eryk Kopczyński. *Half-positional Determinacy of Infinite Games*. PhD thesis, Warsaw University, 2008.
- 32 Alexander Kozachinskiy. Infinite separation between general and chromatic memory. *CoRR*, abs/2208.02691, 2022. doi:10.48550/arXiv.2208.02691.

- 33 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by Boolean combination of winning conditions. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPICs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.FSTTCS.2018.38.
- 34 James C. A. Main and Mickael Randour. Different strokes in randomised strategies: Revisiting Kuhn’s theorem under finite-memory assumptions. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CONCUR.2022.22.
- 35 Oded Maler and Ludwig Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997. doi:10.1016/S0304-3975(96)00312-X.
- 36 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.
- 37 Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993. doi:10.1016/0168-0072(93)90036-D.
- 38 A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958. doi:10.2307/2033204.
- 39 Pierre Ohlmann. Characterizing positionality in games of infinite duration over infinite graphs. In Christel Baier and Dana Fisman, editors, *LICS ’22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 – 5, 2022*, pages 22:1–22:12. ACM, 2022. doi:10.1145/3531130.3532418.
- 40 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.
- 41 Mickael Randour. Automated synthesis of reliable and efficient systems through game theory: A case study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.
- 42 Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.
- 43 Ludwig Staiger. Finite-state omega-languages. *J. Comput. Syst. Sci.*, 27(3):434–448, 1983. doi:10.1016/0022-0000(83)90051-X.
- 44 Klaus Wagner. On ω -regular sets. *Information and control*, 43(2):123–177, 1979.
- 45 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

Expanders in Higher Dimensions

Irit Dinur 

Weizmann Institute of Science, Rehovot, Israel

Abstract

Expander graphs have been studied in many areas of mathematics and in computer science with versatile applications, including coding theory, networking, computational complexity and geometry.

High-dimensional expanders are a generalization that has been studied in recent years and their promise is beginning to bear fruit. In the talk, I will survey some powerful local to global properties of high-dimensional expanders, and describe several interesting applications, ranging from convergence of random walks to construction of locally testable codes that prove the c^3 conjecture (namely, codes with constant rate, constant distance, and constant locality).

2012 ACM Subject Classification Theory of computation → Expander graphs and randomness extractors

Keywords and phrases Expanders

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.4

Category Invited Talk



© Irit Dinur;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 4; pp. 4:1–4:1



Leibniz International Proceedings in Informatics

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

Packing Arc-Disjoint 4-Cycles in Oriented Graphs

Jasine Babu ✉

Indian Institute of Technology Palakkad, India

R. Krithika ✉

Indian Institute of Technology Palakkad, India

Deepak Rajendraprasad ✉

Indian Institute of Technology Palakkad, India

Abstract

Given a directed graph G and a positive integer k , the ARC DISJOINT r -CYCLE PACKING problem asks whether G has k arc-disjoint r -cycles. We show that, for each integer $r \geq 3$, ARC DISJOINT r -CYCLE PACKING is NP-complete on oriented graphs with girth r . When r is even, the same result holds even when the input class is further restricted to be bipartite. On the positive side, focusing on $r = 4$ in oriented graphs, we study the complexity of the problem with respect to two parameterizations: solution size and vertex cover size. For the former, we give a cubic kernel with quadratic number of vertices. This is smaller than the compression size guaranteed by a reduction to the well-known 4-SET PACKING. For the latter, we show fixed-parameter tractability using an unapparent integer linear programming formulation of an equivalent problem.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases arc-disjoint cycles, bipartite digraphs, oriented graphs, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.5

1 Introduction

DISJOINT CYCLE PACKING is a fundamental problem in graph theory and combinatorial optimization. Given a (directed or undirected) graph G and a positive integer k , the objective of the DISJOINT CYCLE PACKING problem is to determine whether G has k (vertex or arc/edge) disjoint cycles. All variants of DISJOINT CYCLE PACKING are NP-complete [3, 13, 23] and therefore have been studied in various algorithmic realms. The fixed-parameter tractable (FPT) algorithm (with respect to the number k of cycles as the parameter) [7] given by Bodlaender in 1994 for VERTEX DISJOINT CYCLE PACKING is one of the earliest results in the parameterized complexity framework. This problem does not admit polynomial kernels [9] but has a lossy kernel [28]. However, EDGE DISJOINT CYCLE PACKING has a polynomial kernel (and hence is also FPT) [9].

While DISJOINT CYCLE PACKING in undirected graphs is amenable to parameterized algorithms, their directed-analogues are not. On directed graphs, VERTEX DISJOINT CYCLE PACKING and ARC DISJOINT CYCLE PACKING are both W[1]-hard [3, 31]. Therefore, studying this problem on a subclass of directed graphs and studying DISJOINT r -CYCLE PACKING (where the length of each cycle in the solution set is required to be r) are natural directions of research. Both VERTEX DISJOINT CYCLE PACKING and ARC DISJOINT CYCLE PACKING are NP-complete but FPT on tournaments [5, 6]. However, these problems are W[1]-hard on bipartite digraphs [3, 31]. In this paper, we focus on ARC DISJOINT r -CYCLE PACKING in oriented graphs. Bessy et. al., have shown that ARC DISJOINT 3-CYCLE PACKING is NP-Complete on tournaments [5]. From this, it follows easily (by reduction via subdivision of arcs) that, for each $q \geq 1$, ARC DISJOINT $3q$ -CYCLE PACKING is NP-complete



© Jasine Babu, R. Krithika, and Deepak Rajendraprasad;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 5; pp. 5:1–5:16



Leibniz International Proceedings in Informatics

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

on oriented graphs with girth $3q$, and that the input class can be restricted to be bipartite when q is even. This leaves open the complexity of ARC DISJOINT r -CYCLE PACKING in digraphs for $r \not\equiv 0 \pmod{3}$. Our first set of results close this gap.

We show that ARC DISJOINT r -CYCLE PACKING is NP-complete on oriented graphs for each integer $r \geq 3$, by a reduction from a variant of SATISFIABILITY [23, LO1].

- (Theorem 1) For each integer $r \geq 3$, ARC-DISJOINT r -CYCLE PACKING on oriented graphs of girth r is NP-complete. Further, for each even integer $r \geq 4$, this result holds even when the input graph is restricted to be bipartite.

It is easy to verify that ARC DISJOINT r -CYCLE PACKING reduces to r -SET PACKING. In r -SET PACKING, given a family \mathcal{F} of sets over a universe U , where each set in the family has cardinality at most r , and a positive integer k , the objective is to decide whether there are sets $S_1, \dots, S_k \in \mathcal{F}$ that are pairwise disjoint. Note that r is fixed. Given an instance (G, k) of ARC DISJOINT r -CYCLE PACKING, the instance $(E(G), \mathcal{C}, k)$ of r -SET PACKING where \mathcal{C} is the set of r -cycles of G is equivalent to it. It is well-known that r -SET PACKING admits a kernel with $\mathcal{O}(k^r)$ sets [17] and $\mathcal{O}(k^{r-1})$ elements [1, 29] leading to a straight-forward $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$ -time algorithm. Further, using the standard color-coding technique [2, 12, 30], r -SET PACKING admits an $\mathcal{O}^*(2^{\mathcal{O}(k)})$ -time algorithm. These results imply that ARC DISJOINT r -CYCLE PACKING in general digraphs admits an FPT algorithm with running time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ and a polynomial kernel. However, the kernel with $\mathcal{O}(k^r)$ sets and $\mathcal{O}(k^{r-1})$ elements for r -SET PACKING does not straightaway give a kernel of same size for ARC DISJOINT r -CYCLE PACKING. Restricting our attention to ARC DISJOINT 4-CYCLE PACKING, we obtain a cubic kernel with quadratic number of vertices.

- (Theorem 2) ARC DISJOINT 4-CYCLE PACKING in oriented graphs has a kernel with $\mathcal{O}(k^3)$ edges and $\mathcal{O}(k^2)$ vertices.

In parameterized complexity, solution size is one of the most natural and almost always the first parameter considered for an optimization problem. Though this parameterization has proven to be fruitful, solution size does not reflect the input structure. Therefore, structural parameters like treewidth, the size of a vertex cover, the size of a feedback vertex set and in general the size of a modulator to a family of graphs have been considered in the literature [11, 12, 16, 18, 24]. In the context of DISJOINT CYCLE PACKING, the parameters that have been studied are treewidth, the size of a vertex cover, the size of a feedback vertex set and a modulator to a cluster graph and max leaf number [8, 24]. The importance in these structural parameters is partly due to their practical relevance and partly due to their role in identifying parameterizations that yield FPT algorithms. In this spirit, we study the complexity of ARC DISJOINT 4-CYCLE PACKING parameterized by the size of a vertex cover (of the underlying undirected graph). We first reduce this problem to ARC DISJOINT 4-CYCLE PACKING in oriented bipartite graphs parameterized by the size ℓ of one of the parts of the bipartition. Then we define a new equivalent problem of building a multidigraph with certain decomposition properties and give an integer linear programming formulation for it where the number of variables is a function of ℓ . Finally, by invoking the FPT algorithm for INTEGER LINEAR PROGRAMMING parameterized by the number of variables [12, 22, 25, 26], we obtain the following result.

- (Theorem 3) ARC DISJOINT 4-CYCLE PACKING in oriented graphs is FPT with respect to size of a vertex cover as the parameter.

Road Map. The paper is organized as follows. In Section 2, we give the necessary definitions related to directed graphs. In Section 3, we show the NP-completeness of ARC DISJOINT r -CYCLE PACKING in oriented (bipartite) graphs of girth r . In Section 4, we describe FPT algorithms and polynomial kernels for ARC DISJOINT 4-CYCLE PACKING. Finally, we conclude with some remarks and open problems in Section 5.

2 Preliminaries

The set $\{1, 2, \dots, n\}$ is denoted by $[n]$. A *multidigraph* is a pair (V, A) consisting of a set V of *vertices* and a multiset A of ordered pairs of vertices (called *arcs*) in V . A *directed graph* (or *digraph*) is a multidigraph (V, A) where A is a set of ordered pairs of distinct vertices in V . Note that a digraph is a multidigraph with no self-loops or multiple/parallel arcs. An arc is specified as an ordered pair of vertices and this pair of vertices are called as its *endpoints*. A *matching* is a collection of arcs that do not share any endpoint. For a digraph G , $V(G)$ and $A(G)$ denote the set of its vertices and the set of its arcs, respectively. An *oriented graph* is a digraph G having no pair of vertices $u, v \in V(G)$ with $(u, v), (v, u) \in A(G)$. A *bipartite (di)graph* is a (di)graph G whose vertex set can be partitioned into two sets X and Y such that every arc/edge in G has one endpoint in X and the other endpoint in Y . We denote such a digraph as $G[X, Y]$ and say that (X, Y) is a bipartition of G .

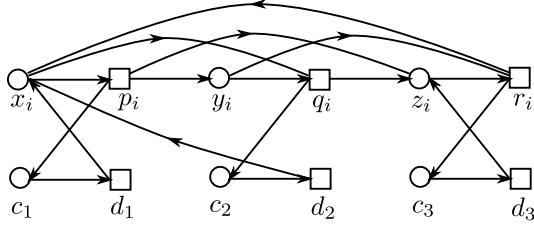
Two vertices u, v are said to be *adjacent* in G if $(u, v) \in A(G)$ or $(v, u) \in A(G)$. For a set of arcs F , $V(F)$ denotes the union of the sets of endpoints of arcs in F . A vertex u is said to be an *in-neighbour* of v if $(u, v) \in A(G)$. Similarly, a vertex u is said to be an *out-neighbour* of v if $(v, u) \in A(G)$. For a vertex $v \in V(G)$, its *out-neighborhood*, denoted by $N^+(v)$, is the set $\{u \in V(G) \mid (v, u) \in A(G)\}$ and its *in-neighborhood*, denoted by $N^-(v)$, is the set $\{u \in V(G) \mid (u, v) \in A(G)\}$. The *out-degree* and *in-degree* of a vertex v are the sizes of its out-neighborhood and in-neighborhood, respectively. For a set $X \subseteq V(G) \cup A(G)$, $G - X$ denotes the digraph obtained from G by deleting X .

A *path* P in G is a sequence (v_1, \dots, v_k) of distinct vertices such that for each $i \in [k - 1]$, $(v_i, v_{i+1}) \in A(G)$. We say that P *starts at* v_1 and *ends at* v_k and also refer to v_1 and v_k as the endpoints of P . The set $\{v_1, \dots, v_k\}$ is denoted by $V(P)$ and the set $\{(v_i, v_{i+1}) \mid i \in [k - 1]\}$ is denoted by $A(P)$. A *cycle* C in G is a sequence (v_1, \dots, v_k) of distinct vertices such that (v_1, \dots, v_k) is a path and $(v_k, v_1) \in A(G)$. The set $\{v_1, \dots, v_k\}$ is denoted by $V(C)$ and the set $\{(v_i, v_{i+1}) \mid i \in [k - 1]\} \cup \{(v_k, v_1)\}$ is denoted by $A(C)$. The length of a path or cycle X is the number of vertices in it. A cycle (path) of length q is called a q -cycle (q -path) and a cycle on three vertices is also called a *triangle*. A collection of q -cycles is called a C_q -*packing* or a q -*cycle packing*.

For details on parameterized algorithms, we refer to standard books in the area [12, 14, 19, 21].

3 NP-Completeness

In this section, we show that for each even integer $r \geq 4$, ARC-DISJOINT r -CYCLE PACKING is NP-complete on oriented bipartite graphs of girth r and for each integer $r \geq 3$, ARC-DISJOINT r -CYCLE PACKING is NP-complete on oriented graphs of girth r . It is easy to verify that ARC DISJOINT r -CYCLE PACKING is in NP. To prove NP-hardness, we give a polynomial-time reduction from a variant of the SATISFIABILITY problem. Let $\text{SAT}(1, 2)$ denote SATISFIABILITY restricted to formulas with at most 3 variables per clause and each variable occurring exactly once negatively and once or twice positively in the formula.



■ **Figure 1** Gadget corresponding to the variable X_i that appears negatively in C_1 and positively in C_2 and C_3 . Here, circles and squares denote a bipartition.

It is well-known that SATISFIABILITY is NP-complete when restricted to instances with 2 or 3 variables per clause and at most 3 occurrences per variable [32, Theorem 2.1]. By a straightforward transformation of instances of this problem into instances of SAT(1, 2), it follows that SAT(1, 2) is also NP-complete.

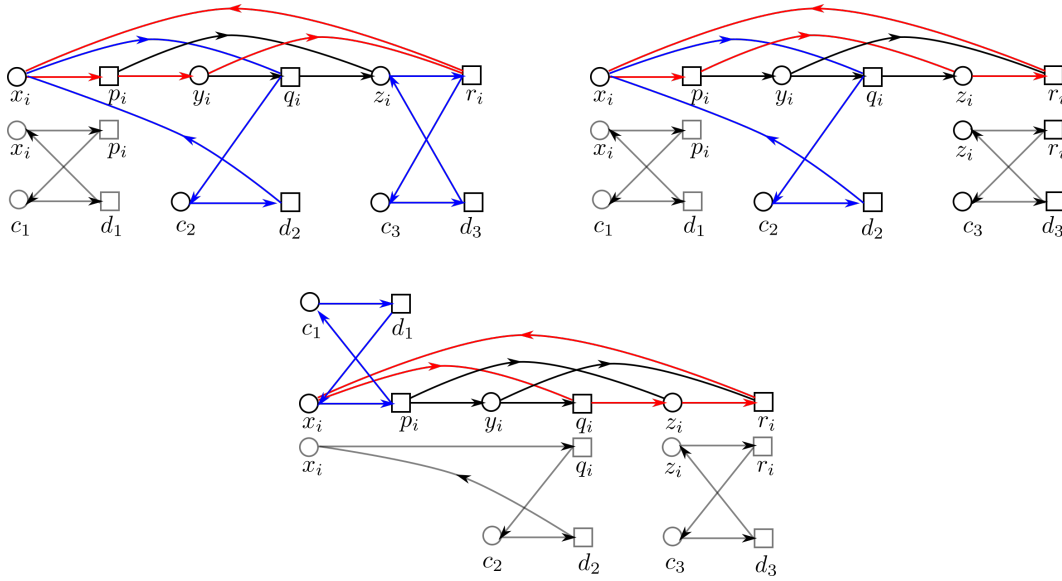
Now, we proceed to the NP-hardness of ARC-DISJOINT r -CYCLE PACKING. First, we show the hardness of ARC-DISJOINT 4-CYCLE PACKING and then move on to the general case. Consider an instance ψ of SAT(1,2) with n variables and m clauses. From ψ , we construct an oriented bipartite graph G such that ψ is satisfiable if and only if G has an arc-disjoint 4-cycle packing of size $m + n$. Let $\{X_1, X_2, \dots, X_n\}$ and $\{C_1, C_2, \dots, C_m\}$ be the sets of variables and clauses, respectively, in ψ . We consider the ordering of the variables (and clauses) given by the increasing order of their indices. The construction of G from ψ is as follows.

- For every $i \in [n]$, add a set of six vertices $\{x_i, y_i, z_i, p_i, q_i, r_i\}$ where $(x_i, p_i, y_i, q_i, z_i, r_i)$ is a directed path and $(r_i, x_i), (x_i, q_i), (y_i, r_i), (p_i, z_i) \in A(G)$.
- For every $j \in [m]$, add two vertices c_j and d_j along with the arc (c_j, d_j) .
- For every $i \in [n]$ and $j \in [m]$ such that X_i appears in C_j ,
 - if the appearance is as a negative literal, then add arcs (d_j, x_i) and (p_i, c_j) .
 - if the appearance is as a positive literal and this is the first such appearance, then add arcs (d_j, x_i) and (q_i, c_j) .
 - if the appearance is as a positive literal and this is the second such appearance, then add arcs $(d_j, z_i), (r_i, c_j)$.

Refer to Figure 1 for an illustration. In order to prove the correctness of the reduction, we first make some observations about the type of 4-cycles in G . Notice that for each clause C_j , the vertices c_j and d_j in G respectively have out-degree one and in-degree one. Hence, any 4-cycle containing one of these vertices must contain the arc (c_j, d_j) . Further, observe that if a 4-cycle contains the arc (c_j, d_j) , then the other two vertices of that cycle must belong to the the same variable gadget X_i for some $i \in [n]$. We will refer to such a cycle as a *clause-variable cycle* involving C_j and X_i . Notice that if there is a clause-variable cycle involving C_j and X_i , then it must be one of the following: (c_j, d_j, x_i, p_i) , (c_j, d_j, x_i, q_i) or (c_j, d_j, z_i, r_i) . The first one will be referred to as the *negative clause-variable cycle* and the other two will be referred to as *positive clause-variable cycles*.

► **Observation 3.1.** *Let $j \in [m]$. If a 4-cycle in G contains a vertex c_j or d_j , then it must be a clause-variable cycle. In any arc-disjoint 4-cycle packing of G , there is at most one clause-variable cycle involving C_j .*

Now, consider a 4-cycle of G that does not contain c_j or d_j vertices for any $j \in [m]$. In any such cycle, all its four vertices must belong to the same variable gadget X_i for some $i \in [n]$. This is because there are no edges between vertices that belong to two distinct variable



■ **Figure 2** The 3 different variable cycles highlighted in red with the clause-variable cycles that are arc-disjoint to it highlighted in blue and the clause-variable cycles that are not arc-disjoint to it shown separately in grey.

gadgets. Any such 4-cycle whose vertex set is contained in the variable gadget of X_i would be referred to as a *variable cycle* of X_i . Notice that the variable cycles of X_i possible are (x_i, p_i, y_i, r_i) , (x_i, p_i, z_i, r_i) and (x_i, q_i, z_i, r_i) . We will call (x_i, p_i, y_i, r_i) as the *first positive variable cycle* of X_i , (x_i, p_i, z_i, r_i) as the *second positive variable cycle* of X_i and (x_i, q_i, z_i, r_i) as the *negative variable cycle* of X_i . Note that all these three cycles contain the arc (r_i, x_i) . Hence, we can make the following observation.

► **Observation 3.2.** Any 4-cycle in G which is not a clause-variable cycle must be a variable cycle. Let $i \in [n]$. In any arc-disjoint 4-cycle packing of G , there is at most one variable cycle of X_i .

► **Observation 3.3.** Let $i \in [n]$.

- The first positive variable cycle of X_i is arc-disjoint from positive clause-variable cycles involving any clause C_j and X_i .
- The negative variable cycle of X_i is arc-disjoint from negative clause-variable cycles involving any clause C_j and X_i .
- If X_i appears positively in clauses C_j and C_k , then the positive clause-variable cycle involving C_j and X_i is arc-disjoint from the positive clause-variable cycle involving C_k and X_i .

The following is another crucial observation.

► **Observation 3.4.** Suppose \mathcal{F} is a family of arc-disjoint 4-cycles in G . Let $i \in [n]$.

- If \mathcal{F} contains a negative clause-variable cycle involving clause C_j and X_i for some $j \in [m]$, then \mathcal{F} cannot contain a (first or second) positive variable cycle of X_i .
- If \mathcal{F} contains a positive clause-variable cycle involving clause C_j and X_i for some $j \in [m]$, then \mathcal{F} cannot contain a negative variable cycle of X_i .

Refer to Figure 2 for an illustration.

► **Lemma 3.1.** *ψ is a YES-instance of SAT(1,2) if and only if $(G, m+n)$ is a YES-instance of ARC-DISJOINT 4-CYCLES.*

Proof. From Observations 3.1 and 3.2, it follows that any arc-disjoint 4-cycle packing of G can contain at most $m+n$ cycles.

Suppose ψ is a YES-instance of SAT(1,2). We will produce a family \mathcal{F} consisting of $m+n$ arc-disjoint 4-cycles in G . Let τ be a satisfying truth assignment of ψ . When ψ is evaluated on τ , in each clause C_j of ψ , at least one literal gets the truth value TRUE. From each clause C_j , choose any one literal l_j which gets the truth value TRUE and do the following.

- if l_j is a positive occurrence of a variable X_i , then add a positive clause-variable cycle involving C_j and X_i to \mathcal{F} .
- if l_j is a negative occurrence of a variable X_i , then add the negative clause-variable cycle involving C_j and X_i to \mathcal{F} .

For each variable X_i do the following.

- if X_i is assigned to be TRUE, add the first positive variable cycle of X_i to \mathcal{F} .
- otherwise, add the negative variable cycle of X_i to \mathcal{F} .

From Observation 3.3, it follows that the cycles added to \mathcal{F} are arc-disjoint. Further, we have added $m+n$ cycles to \mathcal{F} .

To prove the converse, suppose there exist a family \mathcal{F} consisting of $m+n$ arc-disjoint 4-cycles in G . We will produce a truth assignment τ that satisfies ψ . By Observations 3.1 and 3.2, among the cycles in \mathcal{F} , exactly m are clause-variable cycles and exactly n are variable cycles, one corresponding to each variable. If the variable cycle of X_i is a positive variable cycle, assign X_i to be TRUE. If the variable cycle of X_i is a negative variable cycle, assign X_i to be FALSE. This defines the truth assignment τ . Now, it remains to show that τ satisfies ψ . Consider any clause C_j of ψ . Since \mathcal{F} contains exactly m clause-variable cycles, there must be a clause-variable cycle in \mathcal{F} that involves C_j and some variable X_i . By Observation 3.4, if the variable cycle of X_i in \mathcal{F} is a positive variable cycle, then the clause-variable cycle in \mathcal{F} involving C_j and X_i is a positive clause variable cycle. If this happens, then X_i appears positively in C_j . As per our truth assignment τ , we have assigned X_i to be TRUE and hence, the clause C_j will be satisfied. Similarly, from Observation 3.4, we can also show that if variable cycle of X_i in \mathcal{F} is a negative variable cycle, then we would have assigned X_i to FALSE, in which case as well, the clause C_j will be satisfied by τ . Since the same argument holds for every clause C_j , we can see that τ satisfies all clauses of ψ simultaneously. ◀

Lemma 3.1 along with the fact that the reduction described runs in polynomial time leads to the following result.

► **Lemma 3.2.** *ARC-DISJOINT 4-CYCLE PACKING in oriented bipartite graphs is NP-complete.*

Now, we give a generalization of the above reduction by constructing a graph G_r from ψ , for each integer $r \geq 4$. If $r = 4$, then $G_r = G$. If $r > 4$, then we make the following two modifications in G to obtain G_r .

- For every $i \in [n]$, replace the arc (r_i, x_i) in G by a path $P_i = (r_i, a_{i,1}, a_{i,2}, \dots, a_{i,r-4}, x_i)$ of length $r-2$ from r_i to x_i in G_r .
- For every $j \in [m]$, replace the arc (c_j, d_j) in G by a path $Q_j = (c_j, b_{j,1}, b_{j,2}, \dots, b_{j,r-4}, d_j)$ of length $r-2$ from c_j to d_j in G_r .

By extending similar arguments as in the case of G , it can be seen that for any $j \in [m]$ if a cycle in G_r contains either vertex c_j or vertex d_j , then that cycle must contain the entire path Q_j . Further, similar to G , for any cycle of G_r that does not contain c_j or d_j vertices for

any $j \in [m]$, all vertices of the cycle must belong to the same variable gadget X_i for some $i \in [n]$ and the cycle must necessarily contain the entire path P_i . From these arguments, it can be seen that G_r has no cycles of length less than r .

By extending our earlier definitions, r -cycles in G_r that contain Q_j for some $j \in [m]$ will be called as clause-variable cycles and r -cycles in G_r that contain P_i for some $i \in [n]$ will be called as variable cycles. The terminology of negative and positive clause variable cycles can also be generalized in the same manner. It can be seen that a variable cycle that contains P_i has exactly two vertices from outside P_i and they both belong to the gadget of variable X_i itself and a clause-variable cycle that contains Q_j has exactly two vertices from outside Q_j and they both belong to the same variable gadget.

Our remaining arguments for the case when $r = 4$ also extend easily so that the statements of Observation 3.1, Observation 3.2, Observation 3.3, Observation 3.4 and Lemma 3.1 can be generalized by replacing G with G_r and 4 with r . It may be noted that the graph G_r obtained has girth r and when r is even, G_r is bipartite. Further, it is known from the literature that ARC-DISJOINT 3-CYCLE PACKING is NP-complete for tournaments [5]. Hence, we have the following theorem, obtained as a generalization of Lemma 3.2.

► **Theorem 1.** *For each integer $r \geq 3$, ARC-DISJOINT r -CYCLE PACKING on oriented graphs of girth r is NP-complete. Further, for each even integer $r \geq 4$, this result holds even when the input graph is restricted to be bipartite.*

ARC DISJOINT r_{\leq} CYCLE PACKING is a related problem, where the cycles in the packing are required to be of length at most r . For an odd integer $r > 4$, the answer to ARC DISJOINT r_{\leq} CYCLE PACKING on any input oriented bipartite graph is the same as when the cycles in the packing are restricted to be of length at most $r - 1$. This yields the following corollary.

► **Corollary 1.** *For each integer $r \geq 4$, ARC DISJOINT r_{\leq} CYCLE PACKING on oriented bipartite graphs is NP-complete.*

4 FPT Algorithms and Kernels for Packing Arc-Disjoint 4-Cycles

Now, we move on to describing FPT algorithms and polynomial kernels for ARC DISJOINT 4-CYCLE PACKING in oriented graphs. First we consider the standard parameter (solution size) and then proceed to vertex cover size as the parameter.

4.1 Solution Size as Parameter

As mentioned before, ARC DISJOINT 4-CYCLE PACKING reduces to 4-SET PACKING. While the current fastest FPT algorithm for 4-SET PACKING solves ARC DISJOINT 4-CYCLE PACKING in the same time, the kernel for 4-SET PACKING with $\mathcal{O}(k^3)$ elements and $\mathcal{O}(k^4)$ sets does not straight away give a kernel of same size for ARC DISJOINT 4-CYCLE PACKING. Here, we describe an $\mathcal{O}(k^3)$ sized kernel with $\mathcal{O}(k^2)$ vertices. Towards this, we give two kernelization procedures - one that produces a cubic kernel and an other that gives a quadratic vertex kernel. By combining these two procedures, we get the desired kernel. Our algorithm crucially uses reduction rules based on the *new expansion lemma* [20] and *sunflowers* [15, 19, 21].

Consider an instance (G, k) of ARC DISJOINT 4-CYCLE PACKING. The first reduction rule is a standard preprocessing rule.

► **Reduction Rule 4.1.** *If there is a vertex or an edge in G that is not in any 4-cycle, then delete it.*

The correctness of this rule is immediate as no C_4 -packing can contain a vertex or an edge that is not a part of any 4-cycle. The next reduction rule uses the notion of *sunflowers* [15, 19, 21].

► **Definition 4.1.** (*ℓ-sunflower*) An *ℓ-sunflower* in G with core $C \subseteq A(G)$ is a set \mathcal{S} of ℓ 4-cycles such that for any two elements S and S' in \mathcal{S} , we have $A(S) \cap A(S') = C$.

Now, we describe a rule that identifies edges that may be assumed to be in some solution (if the instance is a YES-instance) using sunflowers.

► **Reduction Rule 4.2.** *If there is an edge e in G such that there is a $(4k - 3)$ -sunflower with $\{e\}$ as the core, then delete e from G and decrease k by 1.*

Note that finding a sunflower with core $\{(u, v)\}$ reduces to finding a maximum matching in the auxiliary bipartite graph G_{uv} with bipartition (A, B) defined as follows: $A = N^-(u)$, $B = N^+(v)$ and a vertex $x \in A$ is adjacent to a vertex $y \in B$ if and only if $(y, x) \in A(G)$. Observe that A and B are not necessarily disjoint and we rename B so that $A \cap B = \emptyset$ before finding the matching. Thus, Reduction Rule 4.2 can be applied in polynomial time. The correctness of the rule is justified by the following lemma.

► **Lemma 4.1.** *If Reduction Rule 4.2 is applicable on (G, k) , then (G, k) is a YES-instance if and only if $(G - e, k - 1)$ is a YES-instance.*

Proof. Let \mathcal{S} denote a $(4k - 3)$ -sunflower in G with core $\{e\}$. Since e is in at most one cycle of any 4-cycle packing of G , it is clear that $(G - e, k - 1)$ is a YES-instance, whenever (G, k) is. Conversely, consider a $(k - 1)$ -sized 4-cycle packing \mathcal{F} of $G - e$. Then, $|A(\mathcal{F})| = 4(k - 1)$ and there are at most $4(k - 1)$ cycles in \mathcal{S} that have an edge from $A(\mathcal{F})$. Therefore, there exists a cycle C in the sunflower \mathcal{S} such that $A(C) \cap A(\mathcal{F}) = \emptyset$. Then, $\mathcal{F} \cup \{C\}$ is a k -sized 4-cycle packing of G . ◀

Subsequently, we assume that (G, k) is an instance on which Reductions Rules 4.1 and 4.2 are not applicable. Let \mathcal{X} be a maximal set of 4-cycles in G such that for any two elements X and Y in \mathcal{X} , we have $|A(X) \cap A(Y)| \leq 1$. The following lemma shows that if \mathcal{X} is sufficiently large, then (G, k) is a YES-instance.

► **Lemma 4.2.** *If $|\mathcal{X}| > 16k^2$, then there is a set $\mathcal{C} \subseteq \mathcal{X}$ of k arc-disjoint 4-cycles that can be obtained in polynomial time.*

Proof. Obtain a sequence $\mathcal{F}_1, \dots, \mathcal{F}_r$ of disjoint subsets of \mathcal{X} as follows. For $i \geq 1$, consider an arbitrary 4-cycle $C_i \in \mathcal{X} \setminus \bigcup_{j=1}^{i-1} \mathcal{F}_j$ and let \mathcal{F}_i be the subset of cycles in $\mathcal{X} \setminus \bigcup_{j=1}^{i-1} \mathcal{F}_j$ that share an edge with C_i . Observe that for each $i \in [r]$, $|\mathcal{F}_i| \leq 4(4(k - 1))$ as Reduction Rule 4.2 is not applicable. Further, $r \geq k$ as $|\mathcal{X}| > 16k^2$. Then, C_1, \dots, C_k is the required 4-cycle packing. ◀

Lemma 4.2 lets us apply the following reduction rule.

► **Reduction Rule 4.3.** *If $|\mathcal{X}| > 16k^2$, then replace the instance (G, k) by a constant-sized YES-instance.*

Subsequently, we assume that $|\mathcal{X}| \leq 16k^2$. Define \mathcal{P} to be the set of all 3-paths of G that are in some 4-cycle in \mathcal{X} . Note that \mathcal{P} is $\mathcal{O}(k^2)$. The next lemma says that \mathcal{P} “hits” all 4-cycles in G .

► **Lemma 4.3.** *For every 4-cycle C in G , there is a 3-path $P \in \mathcal{P}$ such that $A(P) \subseteq A(C)$.*

Proof. Consider a 4-cycle C in G . If $C \in \mathcal{X}$, then the claim trivially holds. Otherwise, $C \notin \mathcal{X}$ and by the maximality of \mathcal{X} , there is a 4-cycle X in \mathcal{X} such that $|A(X) \cap A(C)| \geq 2$. Let e and e' be two common arcs of X and C . If e and e' share an endpoint, then the 3-path formed by e and e' that is present in X and C is also in \mathcal{P} and the claim holds. Otherwise, e and e' form a matching implying that $X = C$ leading to a contradiction. \blacktriangleleft

The next rule is one that uses the notion of *expansion* and *new expansion lemma* [20].

► **Definition 4.2** (ℓ -expansion). *Let ℓ be a positive integer and H be a bipartite graph with bipartition (A, B) . For $\hat{A} \subseteq A$ and $\hat{B} \subseteq B$, a set $M \subseteq E(H)$ of edges is called an ℓ -expansion of \hat{A} onto \hat{B} if the following properties hold.*

- every vertex of \hat{A} is incident to exactly ℓ edges in M .
- exactly $\ell|\hat{A}|$ vertices in \hat{B} are incident to edges in M .

For an ℓ -expansion M of \hat{A} onto \hat{B} , we call the vertices of \hat{B} that are endpoints of edges in M as *saturated* and the remaining vertices of \hat{B} as *unsaturated*. Observe that a 1-expansion is simply a matching of \hat{A} to \hat{B} that saturates \hat{A} .

► **Proposition 4.1** (New ℓ -Expansion Lemma, [20]). *Let ℓ be a positive integer and H be a bipartite graph with bipartition (A, B) . Then there exists $\hat{A} \subseteq A$ and $\hat{B} \subseteq B$ such that there is an ℓ -expansion M of \hat{A} onto \hat{B} in H , $N(\hat{B}) \subseteq \hat{A}$ and $|B \setminus \hat{B}| \leq \ell|A \setminus \hat{A}|$. Moreover, the sets \hat{A} and \hat{B} (and M) can be computed in polynomial time.*

Note that \hat{B} (and \hat{A}) may be empty. In that case, since $|B \setminus \hat{B}| \leq \ell|A \setminus \hat{A}|$, we have $|B| \leq \ell|A|$. Therefore, if $|B| > \ell|A|$, then $\hat{B} \neq \emptyset$. Now, we are ready to state the next reduction rule.

We will apply Proposition 4.1 on an auxiliary bipartite graph \hat{G} with bipartition (A, B) where $A = \mathcal{P}$ and $B = V(G) \setminus V(\mathcal{P})$, and a vertex (x, y, z) in A is adjacent to $v \in B$ if and only if (x, y, z, v) is a 4-cycle in G .

► **Reduction Rule 4.4.** *Let $\hat{A} \subseteq A$ and $\hat{B} \subseteq B$ be the sets and M be the ℓ -expansion computed by Proposition 4.1 on \hat{G} for $\ell = 1$. Let $U \subseteq \hat{B}$ be the set of unsaturated vertices of M . Delete U from G .*

Observe that after the application of this rule, if \hat{B} is empty, then $|V(G)| \leq 3|A| + |B| \leq 3|A| + |A|$. Otherwise, \hat{B} is non-empty and $|V(G)| \leq 3|A| + |\hat{B} \setminus U| + |B \setminus \hat{B}| \leq 3|A| + |\hat{A}| + |A \setminus \hat{A}|$. In both cases, $|V(G)|$ is $\mathcal{O}(k^2)$. The correctness of the rule is justified by the following lemma.

► **Lemma 4.4.** *(G, k) is a yes-instance if and only if $(G - U, k)$ is a yes-instance.*

Proof. The “if” part of the claim follows from the fact that $G - U$ is a subgraph of G . Now, suppose (G, k) is a yes-instance. Let \mathcal{C} be a k -sized 4-cycle packing of G . Note that by Lemma 4.3, for each $C \in \mathcal{C}$, there exists a 3-path P_C in \mathcal{P} that is contained in C . Let $\hat{\mathcal{C}}$ be the set of cycles in \mathcal{C} such that for each $C \in \hat{\mathcal{C}}$, there is a 3-path P_C in \hat{A} such that P_C is contained in C . Note that each 4-cycle in $\mathcal{C} \setminus \hat{\mathcal{C}}$ is also a 4-cycle in $G - U$ as $N(\hat{B}) \subseteq \hat{A}$. Let $\hat{\mathcal{C}} = \{C_1, \dots, C_r\}$ and for each $i \in [r]$, let $P_i = (x_i, y_i, z_i)$ be a 3-path in \hat{A} that is contained in C_i . As \hat{A} is saturated by a matching M in \hat{G} , there is a vertex $v_i \in B$ matched to the vertex P_i in \hat{A} . By replacing each C_i by (x_i, y_i, z_i, v_i) in \mathcal{C} , we get a k -sized 4-cycle packing of $G - U$. \blacktriangleleft

Now, we have the following property on an instance on which none of the reduction rules described so far is applicable.

5:10 Packing Arc-Disjoint 4-Cycles in Oriented Graphs

► **Proposition 4.2.** *If (G, k) is an instance of ARC DISJOINT 4-CYCLE PACKING on which none of the Reduction Rules 4.1, 4.2, 4.3, 4.4 is applicable, then G has $\mathcal{O}(k^2)$ vertices.*

Note that since all the reduction rules described can be applied in polynomial time, Proposition 4.2 gives us a quadratic vertex kernel for ARC DISJOINT 4-CYCLE PACKING. However, this kernel may have $\mathcal{O}(k^4)$ edges.

Next, we describe a procedure that reduces the number of edges to $\mathcal{O}(k^3)$. This procedure is a stand-alone algorithm that produces a cubic kernel. We begin with the definition of C_4 -partners.

► **Definition 4.3.** *Two arcs (x, y) and (z, v) are called C_4 -partners in G if (x, y, z, v) is a C_4 in G .*

For an arc (x, y) in G , E_{xy} denotes the set of all C_4 -partners of (x, y) . We will rightfully treat E_{xy} as a subdigraph of G . For a collection of subgraphs \mathcal{H} of G , $A(\mathcal{H})$ denotes $\bigcup_{H \in \mathcal{H}} A(H)$. For a set S and a natural number q , $\lfloor S \rfloor_q$ denotes S itself if $|S| \leq q$ and an arbitrary q -sized subset of S otherwise.

We now describe a procedure that marks a certain number of 4-cycles and takes the subgraph spanned by the arcs of the marked cycles as the kernel. This marking procedure is described as Algorithm 1.

■ **Algorithm 1** MarkingProcedure.

Require: A digraph G and a positive integer k

Ensure: A subgraph H of G on $\mathcal{O}(k^3)$ edges such that H has an arc-disjoint C_4 -packing of size k if and only if G has an arc-disjoint C_4 -packing of size k .

Find a maximal arc-disjoint C_4 -packing \mathcal{P} in G

if $|\mathcal{P}| \geq k$ **then**

$A(H) \leftarrow A(\mathcal{P}')$ and $V(H) \leftarrow V(\mathcal{P}')$, where \mathcal{P}' is any subset of k distinct cycles from \mathcal{P} .

return H

for each arc $(x, y) \in A(\mathcal{P})$ **do**

if E_{xy} has a matching M of size $4k$ **then**

$F_{xy} = M$

else

Let S be the set of endpoints of a maximum matching in E_{xy}

$F_{xy} \leftarrow \emptyset$

for each vertex $s \in S$ **do**

Let A_s be the set of arcs in E_{xy} outgoing from s

Let B_s be the set of arcs in E_{xy} incoming to s

Add $\lfloor A_s \rfloor_{4k} \cup \lfloor B_s \rfloor_{4k}$ to F_{xy}

for each arc $(z, v) \in F_{xy}$ **do**

mark the 4-cycle (x, y, z, v)

Let H be the subgraph of G whose arc set is the union of the arcs of all the marked 4-cycles and vertex set is the set of endpoints of its arcs.

return H

We show the correctness of **MarkingProcedure** in the following lemma.

► **Lemma 4.5.** *Given a digraph G and a positive integer k , **MarkingProcedure** (Algorithm 1) returns a digraph H on $\mathcal{O}(k^3)$ edges and $\mathcal{O}(|V(G)|)$ vertices such that H has an arc-disjoint C_4 -packing of size k if and only if G has an arc-disjoint C_4 -packing of size k .*

Proof. Since H is a subgraph of G , the “only if” direction is immediate. To prove the other direction, suppose (G, k) is a yes-instance and let $\mathcal{Q} = \{Q_1, \dots, Q_k\}$ be an arc-disjoint C_4 -packing in G which has the maximum number of edges from H . If all the cycles in \mathcal{Q} are present in H , then it has a k -sized packing as required. Hence for the rest of the proof we assume Q_1 is not contained in H and show that we can replace Q_1 with another 4-cycle Q'_1 which is contained in H and arc-disjoint from Q_2, \dots, Q_k . Then the C_4 -packing $\{Q'_1, Q_2, \dots, Q_k\}$ contradicts the choice of \mathcal{Q} .

Since Q_1 is a 4-cycle in G , and \mathcal{P} is a maximal C_4 -packing in G , there is an arc (x, y) of Q_1 present in $A(\mathcal{P})$. Let $Q_1 = (x, y, z, v)$. If (z, v) was added to F_{xy} by the algorithm, Q_1 would have been marked and hence Q_1 would be present in H . Hence we assume that $(z, v) \notin F_{xy}$. If E_{xy} had a matching of size $4k$, then there are $4k$ arc-disjoint 3-length paths from y to x in H . At most $4(k - 1)$ of them can be hit by other cycles Q_2, \dots, Q_k of the packing \mathcal{Q} . Let (y, z', v', x) be a 3-length path in H that is arc-disjoint from Q_2, \dots, Q_k . We can replace $Q_1 = (x, y, z, v)$ with $Q'_1 = (x, y, z', v')$ in \mathcal{Q} .

Suppose E_{xy} did not have a matching of size $4k$. Then either z or v (or both) belong to S . We will argue the case when $z \in S$. The case $v \in S$ is similar. Since (z, v) is not in F_{xy} , the set F_{xy} contains $4k$ other edges of E_{xy} outgoing from z . Hence there are $4k$ arc-disjoint 2-length paths from z to x in H . At most $4(k - 1)$ of them can be hit by other cycles $Q_2 \dots Q_k$ of the packing \mathcal{Q} . Let (z, v', x) be a 2-length path in H that is arc-disjoint from Q_2, \dots, Q_k . We can replace $Q_1 = (x, y, z, v)$ with $Q'_1 = (x, y, z, v')$ in \mathcal{Q} .

We now prove a bound on the size of H . Observe that for each arc $(x, y) \in A(\mathcal{P})$, $|F_{xy}|$ is bounded above by $4k$ in the first case and $8k|S|$ in the second case. Since $|S| \leq 2(4k - 1)$, we have $|F_{xy}| \leq 64k^2$. Since $|A(\mathcal{P})| < 4k$, the total number of 4-cycles that are marked is less than $256k^3$ and hence H has at most $1024k^3$ arcs. ◀

Applying Algorithm 1 on the instance guaranteed by Proposition 4.2, leads to the following result.

► **Theorem 2.** *ARC DISJOINT 4-CYCLE PACKING admits an $\mathcal{O}(k^3)$ sized kernel with $\mathcal{O}(k^2)$ vertices.*

We remark that in Theorem 2, our focus was on only getting a cubic kernel with quadratic number of vertices and we have not attempted to optimize the constants involved.

4.2 Vertex Cover Size as Parameter

In this section, we study the time complexity of ARC-DISJOINT 4-CYCLE PACKING in oriented graphs parameterized by the size of a vertex-cover. The first step is to reduce this problem to ARC DISJOINT 4-CYCLE PACKING in oriented bipartite graphs parameterized by the size of the smaller part. Then we invent a new problem which is in bijective correspondence with arc-disjoint 4-cycle packing in oriented bipartite graphs. Given an oriented bipartite graph G , we consider the problem of building an auxiliary multidigraph H with vertex set as the smaller part of G which can be decomposed into a disjoint union of a collection of transitive bipartite multidigraphs, each with an upper bound on its chromatic index. We formulate this problem as an INTEGER LINEAR PROGRAMMING problem with its number of variables and constraints, though exponential in the size of the smaller part of G , is independent of the size of the larger part. Hence solving this integer linear program gives an FPT algorithm for the original problem.

► **Lemma 4.6.** *Let G be an oriented graph with a vertex-cover X . G has a collection of k pairwise arc-disjoint 4-cycles if and only if there exists a collection \mathcal{P} of pairwise arc-disjoint 3-paths in $G[X]$ such that the bipartite digraph $G_{\mathcal{P}}$ obtained from G by deleting all the edges inside X and adding for each $P = (a, b, c)$ in \mathcal{P} a new 3-path (a, b_P, c) (where b_P is a new two-degree vertex) contains k pairwise arc-disjoint 4-cycles.*

Proof. Suppose G has a collection \mathcal{C} of k pairwise arc-disjoint 4-cycles. Since X is a vertex-cover of G , for each $C = (a, b, c, d) \in \mathcal{C}$, the intersection of C with $G[X]$ is either the entire C , a 3-path in C , or two vertices of C . In the first case, add (a, b, c) and (c, d, a) to \mathcal{P} . In the second case, add the 3-path in $C \cap G[X]$ to \mathcal{P} . In the third case, the two vertices of C from X are non-adjacent and no path is added to \mathcal{P} . By the arc-disjointness of \mathcal{C} and the above construction, \mathcal{P} is a collection of pairwise arc-disjoint 3-paths in $G[X]$. In this case it is easy to see that the bipartite digraph $G_{\mathcal{P}}$ constructed for \mathcal{P} also contains k pairwise arc-disjoint 4-cycles.

In the other direction, let \mathcal{P} be a collection of pairwise arc-disjoint 3-paths in $G[X]$ such that the bipartite digraph $G_{\mathcal{P}}$ constructed based on it has a family \mathcal{C} of k pairwise arc-disjoint 4-cycles. Since $G_{\mathcal{P}}$ is bipartite with X as one part, every cycle $C = (a, b, c, d)$ in \mathcal{C} has two non-adjacent vertices, say a and c in X and the other two vertices outside X . This splits C into two 3-length paths (a, b, c) and (c, d, a) . Let $\mathcal{P}_{\mathcal{C}}$ denote this collection of $2k$ pairwise arc-disjoint paths obtained from the k cycles in \mathcal{C} , each starting and ending in X . We will show that each 3-path in $\mathcal{P}_{\mathcal{C}}$ is either present in G , or it can be substituted by a 3-path in G with the same endpoints, such that the resulting collection $\mathcal{P}'_{\mathcal{C}}$ is pairwise arc-disjoint in G . This would suffice to construct k pairwise arc-disjoint 4-cycles in G . Let (a, b, c) be a 3-path in $\mathcal{P}_{\mathcal{C}}$. If b is a vertex in G , then the same path (a, b, c) is available in G as well and is added to $\mathcal{P}'_{\mathcal{C}}$. If b is a new vertex in $G_{\mathcal{P}}$ but not in G , then b represented a 3-path (a, b', c) in $G[X]$, Add (a, b', c) to $\mathcal{P}'_{\mathcal{C}}$ instead of (a, b, c) . Note that the arc-disjointness among the newly added 3-paths is due to the arc-disjointness in \mathcal{P} and their arc-disjointness with the 3-paths from $\mathcal{P}_{\mathcal{C}}$ is since the former is made up of arcs from $G[X]$, while the latter is made up of arcs across X and $V(G) \setminus X$. ◀

If X has ℓ vertices, then there are at most ℓ^3 3-paths in $G[X]$ and any collection \mathcal{P} of pairwise arc-disjoint 3-paths has at most $\ell^2/2$ paths in it. Hence the number of choices for \mathcal{P} is upper bounded by $\binom{\ell^3}{\ell^2/2} \leq (2e\ell)^{\ell^2/2}$. Hence one can use an algorithm to solve ARC DISJOINT 4-CYCLE PACKING in bipartite graphs to solve the same problem for G with a blow-up of $(2e\ell)^{\ell^2/2}$ in running time.

This leads us to the study of parameterized complexity of ARC DISJOINT 4-CYCLE PACKING in bipartite digraphs G with bipartition $L \cup R$ with respect to $\min\{|L|, |R|\}$ as the parameter. Consider an instance $\mathcal{I} = (G, k)$ with $|L| = \min\{|L|, |R|\} = \ell$. Let $L = \{v_1, \dots, v_{\ell}\}$ and $R = \{u_1, \dots, u_n\}$.

► **Definition 4.4. (Signature)** *For a vertex $u \in R$, the signature of u is a string $\pi_u = (\pi_u(1), \dots, \pi_u(\ell))$ in $\{1, -1, 0\}^{\ell}$ defined as follows.*

$$\pi_u(i) = \begin{cases} 1, & \text{if } u \in N^+(v_i) \\ -1, & \text{if } u \in N^-(v_i) \\ 0, & \text{otherwise} \end{cases}$$

Observe that the signature of vertices in R can be determined in $O(n)$ time and there are at most 3^{ℓ} distinct signatures. Let \mathcal{S} denote the set of possible signatures. For each $\sigma \in \mathcal{S}$, let R_{σ} denote the set of vertices of R that have signature equal to σ . For each $\sigma \in \mathcal{S}$, let $\sigma^{-1}(+1)$ denote the set $\{v_i \in L \mid \sigma(i) = 1\}$ and $\sigma^{-1}(-1)$ denote the set $\{v_i \in L \mid \sigma(i) = -1\}$.

Given two multidigraphs $G_1 = (V, A_1)$ and $G_2 = (V, A_2)$ on the same vertex set, we denote by $G_1 \uplus G_2$ the multidigraph obtained by taking the union of G_1 and G_2 after renaming the arcs in A_2 so that A_2 is disjoint from A_1 . We call $G_1 \uplus G_2$ as the *disjoint union* of G_1 and G_2 and extend the terminology to any finite collection of graphs on a common vertex set. A multidigraph in which for each pair of vertices u, v the number of arcs directed from u to v is the same as the number of arcs directed from v to u is called *balanced*.

► **Lemma 4.7.** *G has a collection of k pairwise arc-disjoint 4-cycles if and only if there exists a collection of bipartite multidigraphs H_σ , $\sigma \in S$, on the vertex set L such that*

- (i) *each arc of H_σ is directed from $\sigma^{-1}(+1)$ to $\sigma^{-1}(-1)$,*
- (ii) *the edges of H_σ can be properly coloured using $|R_\sigma|$ colours, and*
- (iii) *the disjoint union $H = \uplus_{\sigma \in S} H_\sigma$ is a balanced multidigraph with $2k$ arcs.*

Proof. Suppose G contains an arc-disjoint collection \mathcal{C} of 4-cycles, $|\mathcal{C}| = k$. For each $\sigma \in S$, we set $H_\sigma = (L, A_\sigma)$ where A_σ contains an arc from u to w for each $v \in R_\sigma$ such that (u, v, w) is a path in one of the cycles in \mathcal{C} . The first condition in the lemma follows from the definition of signature. The second condition follows since for each $v \in R_\sigma$ the collection of arcs (u, w) in H_σ where (u, v, w) is a path in one of the cycles in \mathcal{C} forms a matching in H_σ . The third condition follows since each 4-cycle (u, v, w, x) in \mathcal{C} (with $u, w \in L$) contributes two opposite arcs, specifically (u, w) and (w, u) , to H .

In the opposite direction, suppose we have a collection of bipartite multigraphs H_σ , $\sigma \in S$, on the vertex set L , satisfying the three conditions of the lemma. We first construct a collection \mathcal{P} of pairwise arc-disjoint P_3 's in G starting and ending in L as follows. Condition 2 assures that we can decompose the arc-set of each H_σ into at most $|R_\sigma|$ matchings H_v , $v \in R_\sigma$. For each arc (u, w) in H_v , add the directed path (u, v, w) to \mathcal{P} . The first condition in the lemma ensures that (u, v, w) is indeed a path in G . Two paths resulting from the same matching H_v , for each $v \in R$, are arc-disjoint because of the disjointness of the end-vertices of the two paths. Two paths resulting from H_v and $H_{v'}$, for each $v, v' \in R, v \neq v'$, are arc-disjoint since their middle vertices are two distinct vertices in R . Hence \mathcal{P} is a collection of arc-disjoint P_3 's from G . Condition 3 guarantees that $|\mathcal{P}| = 2k$ and that the P_3 's in \mathcal{P} can be perfectly paired to form a collection \mathcal{C} of pairwise arc-disjoint 4-cycles in G , with $|\mathcal{C}| = k$. ◀

The feasibility of finding a collection of bipartite multidigraphs satisfying the conditions of Lemma 4.7 can be checked by the following integer linear program.

Set of Variables: $X = \{x_{\sigma,u,w} \mid \sigma \in S, u \in \sigma^{-1}(-1), w \in \sigma^{-1}(+1)\}$

Feasible Solution: An integral assignment to the variables satisfying the following properties.

- For each signature $\sigma \in S$ and $u, w \in L$

$$\sum_{w': x_{\sigma,u,w'} \in X} x_{\sigma,u,w'} \leq |R_\sigma| \text{ and } \sum_{u': x_{\sigma,u',w} \in X} x_{\sigma,u',w} \leq |R_\sigma|$$
- For each pair $u, w \in L$

$$\sum_{\sigma: x_{\sigma,u,w} \in X} x_{\sigma,u,w} = \sum_{\sigma: x_{\sigma,w,u} \in X} x_{\sigma,w,u}$$

Optimum Solution: A feasible solution that maximizes $\sum_{x_{\sigma,u,w} \in X} x_{\sigma,u,w}$.

Since the edge-chromatic number of bipartite multigraphs is equal to their maximum degree [4], the second condition of Lemma 4.7 is equivalent to ensuring that H_σ has a maximum degree at most $|R_\sigma|$. This is ensured by the first group of constraints. The second group of constraints ensure that the disjoint union H is balanced. The edge-count of H is the cost function to be maximized.

INTEGER LINEAR PROGRAMMING is FPT when parameterized by the number of variables by the following result.

► **Proposition 4.3** ([12, 22, 25, 26]). *An integer linear program of size L with p variables can be solved in $\mathcal{O}(p^{2.5p+o(p)}(L + \log M_x) \log(M_x M_c))$ time where M_x is an upper bound on the absolute values a variable can take in a solution and M_c is the largest absolute value of a coefficient in the vector c corresponding to the objective function.*

In our integer linear program, the number of variables p is equal to $|X|$ which is $O(3^\ell \ell^2)$ and the number of constraints is $O(3^\ell \ell)$. So the size L of the integer linear program is $O(3^{2\ell} \ell^3)$. Moreover the maximum value M_x that a variable can take is bounded by $|R_\sigma|$ which is $O(n)$ and all the coefficients are 1 or 0. Hence, Proposition 4.3 gives the following result.

► **Theorem 3.** *ARC DISJOINT 4-CYCLE PACKING parameterized by the size of a vertex cover is FPT.*

5 Concluding Remarks

In this work, we studied ARC DISJOINT r -CYCLE PACKING and showed that it is NP-complete on oriented graphs with girth r and remains so for even r when the input is further restricted to be bipartite. For $r = 4$, we gave a cubic kernel (containing a quadratic number of vertices) with respect to the number of cycles as the parameter and showed fixed-parameter tractability with respect to the size of a vertex cover as the parameter. Improving the size of this kernel or showing tightness and giving an FPT algorithm for the problem parameterized by treewidth are interesting future directions. Note that treewidth of a graph is at most the size of its vertex cover. Also, coming up with the best possible polynomial kernels possible for $r > 4$ is a natural next question.

Tournaments and bipartite tournaments are well-studied special classes of digraphs with interesting structural and algorithmic properties. While ARC DISJOINT CYCLE PACKING is known to be NP-complete in tournaments [5], the complexity of this problem in bipartite tournaments is still open. Further, ARC DISJOINT 4-CYCLE PACKING is also open in bipartite tournaments. The NP-completeness of ARC DISJOINT CYCLE PACKING in tournaments was established by a reduction from SAT(1,2) to ARC DISJOINT 3-CYCLE PACKING in tournaments. The construction of the graph in the reduced instance may be viewed as consisting of two phases where in the first phase, an oriented graph is constructed and in the second phase, this graph is completed into a tournament using a decomposition of the edges of a complete (undirected) graph into triangles [27]. The second phase of this approach does not seem to extend to bipartite tournaments. Further, ARC DISJOINT 4-CYCLE PACKING in bipartite tournaments is closely related to a conjecture by Brualdi and Shen [10] that asserts that every Eulerian bipartite tournament has a decomposition of its arcs into 4-cycles. We believe that resolving the complexity of this problem would shed some light on this conjecture.

References

- 1 Faisal N. Abu-Khazam. An improved kernelization algorithm for r-set packing. *Inf. Process. Lett.*, 110(16):621–624, 2010. doi:10.1016/j.ip1.2010.04.020.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 3 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs – Theory, Algorithms and Applications, Second Edition*. Springer Monographs in Mathematics. Springer, 2009.
- 4 Claude Berge. *Graphs and hypergraphs*. North-Holland Pub. Co., 1973.
- 5 Stéphane Bessy, Marin Bougeret, R. Krithika, Abhishek Sahu, Saket Saurabh, Jocelyn Thiebaut, and Meirav Zehavi. Packing arc-disjoint cycles in tournaments. *Algorithmica*, 83(5):1393–1420, 2021. doi:10.1007/s00453-020-00788-2.
- 6 Stéphane Bessy, Marin Bougeret, and Jocelyn Thiebaut. Triangle packing in (sparse) tournaments: Approximation and kernelization. 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 14:1–14:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.14.
- 7 Hans L. Bodlaender. On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994. doi:10.1142/S0129054194000049.
- 8 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theor. Comput. Sci.*, 511:117–136, 2013. doi:10.1016/j.tcs.2012.09.006.
- 9 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.
- 10 Richard A. Brualdi and Jian Shen. Disjoint cycles in eulerian digraphs and the diameter of interchange graphs. *J. Comb. Theory, Ser. B*, 85(2):189–196, 2002. doi:10.1006/jctb.2001.2094.
- 11 Leizhen Cai. Parameterized complexity of vertex colouring. *Discret. Appl. Math.*, 127(3):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.
- 12 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.
- 13 Dorit Dor and Michael Tarsi. Graph decomposition is NPC – A complete proof of Holyer’s conjecture. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 252–263. ACM, 1992. doi:10.1145/129712.129737.
- 14 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.
- 15 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 1960. doi:10.1112/jlms/s1-35.1.85.
- 16 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013. doi:10.1016/j.ejc.2012.04.008.
- 17 Michael R. Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Ulrike Stege, Dimitrios M. Thilikos, and Sue Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. *Algorithmica*, 52(2):167–176, 2008. doi:10.1007/s00453-007-9146-y.
- 18 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009. doi:10.1007/s00224-009-9167-9.
- 19 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.

- 20 Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019. doi:10.1145/3293466.
- 21 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 22 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 23 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 24 Bart M. P. Jansen. *The Power of Data Reduction: Kernels for Fundamental Graph Problems*. PhD thesis, Utrecht University, The Netherlands, 2013.
- 25 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.
- 26 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.
- 27 Thomas P. Kirkman. On a Problem in Combinations. *Cambridge and Dublin Mathematical Journal*, 2:191–204, 1847.
- 28 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.
- 29 Hannes Moser. *Finding optimal solutions for covering and matching problems*. PhD thesis, Friedrich Schiller University of Jena, 2010. URL: <https://d-nb.info/999819399>.
- 30 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.
- 31 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discret. Math.*, 24(1):146–157, 2010. doi:10.1137/070697781.
- 32 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discret. Appl. Math.*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

Approximate Representation of Symmetric Submodular Functions via Hypergraph Cut Functions

Calvin Beideman  



University of Illinois, Urbana-Champaign, IL, USA

Karthekeyan Chandrasekaran  

University of Illinois, Urbana-Champaign, USA

Chandra Chekuri  

University of Illinois, Urbana-Champaign, USA

Chao Xu  

University of Electronic Science and Technology of China, Chengdu, China

Abstract

Submodular functions are fundamental to combinatorial optimization. Many interesting problems can be formulated as special cases of problems involving submodular functions. In this work, we consider the problem of approximating symmetric submodular functions everywhere using hypergraph cut functions. Devanur, Dughmi, Schwartz, Sharma, and Singh [5] showed that symmetric submodular functions over n -element ground sets cannot be approximated within $(n/8)$ -factor using a graph cut function and raised the question of approximating them using hypergraph cut functions. Our main result is that there exist symmetric submodular functions over n -element ground sets that cannot be approximated within a $o(n^{1/3}/\log^2 n)$ -factor using a hypergraph cut function. On the positive side, we show that symmetrized concave linear functions and symmetrized rank functions of uniform matroids and partition matroids can be constant-approximated using hypergraph cut functions.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Submodular Functions, Hypergraphs, Approximation, Representation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.6

Funding Calvin Beideman: supported in part by NSF grants CCF-1814613 and CCF-1907937.

Karthekeyan Chandrasekaran: supported in part by NSF grants CCF-1814613 and CCF-1907937.

Chandra Chekuri: supported in part by NSF grant CCF-1907937.

1 Introduction

A set function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ defined over a ground set V is *submodular* if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$ and is *symmetric* if $f(A) = f(V - A)$ for all subsets $A \subseteq V$. Submodular functions have the diminishing marginal returns property which arise frequently in economic and game theoretic contexts. Well-known examples of submodular functions include matroid rank functions and graph/hypergraph cut functions. Owing to these connections, submodular functions play a fundamental role in combinatorial optimization.

Throughout this work, we will be interested in non-negative set functions $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ with $f(\emptyset) = 0$. We use n to denote the size of the ground set V . For a parameter $\alpha \geq 1$, a set function $g : 2^V \rightarrow \mathbb{R}_{\geq 0}$ is said to α -approximate a set function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ if

$$g(A) \leq f(A) \leq \alpha g(A) \quad \forall A \subseteq V.$$

Given the prevalence of submodular functions in combinatorial optimization, a natural question that has been studied is whether an arbitrary submodular set function can be well-approximated by a concisely representable function. We distinguish between structural



© Calvin Beideman, Karthekeyan Chandrasekaran, Chandra Chekuri, and Chao Xu; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 6; pp. 6:1–6:18



Leibniz International Proceedings in Informatics

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

and algorithmic variants of this question: the structural question asks whether submodular functions can be well-approximated via concisely representable functions while the algorithmic question asks whether such a concise representation can be constructed using polynomial number of function evaluation queries (note that the algorithmic question is concerned with the number of function evaluation queries as opposed to run-time). Concise representations with small-approximation factor are useful in learning, testing, streaming, and sketching algorithms. Consequently, concise representations with small-approximation factor for submodular functions (and their generalizations and subfamilies of submodular functions) have been studied from all these perspectives with most results focusing on monotone submodular functions [8, 2, 12, 1, 5, 11, 6, 3].

In this work, we focus on approximating *symmetric* submodular functions. Balcan, Harvey, and Iwata [2] showed that for every symmetric submodular function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$, there exists a function $g : 2^V \rightarrow \mathbb{R}_{\geq 0}$ defined by $g(S) := \sqrt{\chi(S)^T M \chi(S)}$, where $\chi(S) \in \{0, 1\}^V$ is the indicator vector of $S \subseteq V$ and M is a symmetric positive definite matrix such that g \sqrt{n} -approximates f . We note that such a function g has a concise representation – namely, the matrix M . Is it possible to improve on the approximation factor for symmetric submodular functions using other concisely representable functions?

The concisely representable family of functions that we study in this work is the family of hypergraph cut functions. A hypergraph $H = (V, E)$ consists of a vertex set V and hyperedges E where each hyperedge $e \in E$ is a subset of vertices. If every hyperedge has size 2, then the hypergraph is simply a graph. For a subset A of vertices, we use $\delta(A)$ to denote the set of hyperedges e such that e has non-empty intersection with both A and $V \setminus A$. The cut function $d : 2^V \rightarrow \mathbb{R}_+$ of a hypergraph $H = (V, E)$ with hyperedge weights $w : E \rightarrow \mathbb{R}_+$ is given by

$$d(A) := \sum_{e \in E: e \cap \delta(A)} w_e \quad \forall A \subseteq V.$$

A function $g : 2^V \rightarrow \mathbb{R}_+$ is a hypergraph cut function if there exists a weighted hypergraph with vertex set V whose cut function is g . We will say that a function $f : 2^V \rightarrow \mathbb{R}_+$ is α -hypergraph-approximable (α -graph approximable) if there exists a hypergraph (graph) cut function g such that g α -approximates f . We note that although a hypergraph could have exponential number of hyperedges, every n -vertex hypergraph admits a $(1 + \epsilon)$ -approximate cut-sparsifier with $O(\frac{n \log n}{\epsilon^2})$ hyperedges (see Theorem 6 for a formal definition of cut-sparsifier), and hence, hypergraph cut functions have a concise representation (with a constant loss in approximation factor).

The structural approximation question of whether every symmetric submodular function is constant-hypergraph-approximable was raised by Devanur, Dughmi, Shwartz, Sharma, and Singh [5]. They showed that every symmetric submodular function on a ground set of size n is $O(n)$ -graph-approximable and that this factor is tight for graph-approximability: in fact, the cut function of the n -vertex hypergraph containing a single hyperedge that contains all vertices cannot be $(n/4 - \epsilon)$ -approximated by a graph cut function for all constant $\epsilon > 0$. This example naturally raises the following intriguing conjecture:

► **Conjecture 1.** *Every symmetric submodular function is constant-hypergraph-approximable.*

The conjecture is further fueled by the fact that there are no natural examples of symmetric submodular functions besides hypergraph cut functions (although arbitrary submodular functions can be symmetrized while preserving submodularity).

We emphasize that the algorithmic variant of Conjecture 1 is false. In particular, there does not exist an algorithm that makes a polynomial number of function evaluation queries to a symmetric submodular function f and constructs a hypergraph cut function g such that g $O(\sqrt{n/\ln n})$ -approximates f . We outline a proof of this observation now. Suppose that there exists an algorithm that uses polynomial number of function evaluation queries to a given symmetric submodular function f to construct a weighted hypergraph whose cut function α -approximates f ; then we can obtain an α -approximation to the *symmetric submodular sparsest cut problem* by constructing such a hypergraph and solving the sparsest cut on that hypergraph exactly (using exponential run-time). However, Svitkina and Fleischer [12] have shown that the best possible approximation for the symmetric submodular sparsest cut problem using polynomial number of function evaluation queries is $\Omega(\sqrt{n/\ln n})$ (even if exponential run-time is allowed). Hence, the algorithmic version of hypergraph-approximability has a strong lower bound of $\Omega(\sqrt{n/\ln n})$. This leaves the structural question open while perhaps, hinting that it may also have a strong lower bound.

1.1 Our Results

The symmetrization of a set function $f : 2^V \rightarrow \mathbb{R}$ is the function $f_{\text{sym}} : 2^V \rightarrow \mathbb{R}$ obtained as

$$f_{\text{sym}}(A) := f(A) + f(V \setminus A) - f(V) - f(\emptyset).$$

We note that if $f : 2^V \rightarrow \mathbb{R}$ is submodular, then its symmetrization $f_{\text{sym}} : 2^V \rightarrow \mathbb{R}$ is symmetric submodular. A matroid rank function is a non-negative integer valued submodular set function $r : 2^V \rightarrow \mathbb{Z}$ satisfying $r(A) \leq r(A \cup \{e\}) \leq r(A) + 1$ for every subset $A \subseteq V$ and element $e \in V$. As a step towards understanding Conjecture 1, we observe that it suffices to focus on symmetrized matroid rank functions (see Section 1.3 for a proof).

► **Proposition 2.** *If the symmetrization of every matroid rank function is α -hypergraph-approximable, then every rational-valued symmetric submodular function is α -hypergraph-approximable.*

Next, we refute Conjecture 1 by showing the following result.

► **Theorem 3.** *For every sufficiently large positive integer n , there exists a matroid rank function $r : 2^{[n]} \rightarrow \mathbb{Z}_{\geq 0}$ such that r_{sym} is not α -hypergraph-approximable for*

$$\alpha = o\left(\frac{n^{\frac{1}{3}}}{\log^2 n}\right).$$

Our proof of Theorem 3 is an existential argument and it does not construct an explicit matroid rank function that achieves the lower bound.

Next, we prove positive approximation results for certain subfamilies of symmetric submodular functions. The subfamilies that we consider are inspired by Proposition 2 and by previous work on approximating symmetric submodular functions and matroid rank functions.

We call a set function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ as a *concave linear function* if there exist weights $w : V \rightarrow \mathbb{R}_{\geq 0}$ and an increasing concave function $h : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ such that $f(S) = h(\sum_{v \in S} w_v)$ for every $S \subseteq V$. We note that concave linear functions are submodular. Goemans, Harvey, Iwata, and Mirrokni [8] showed that every matroid rank function over a n -element ground set can be \sqrt{n} -approximated by the square-root of a linear function, i.e., by a concave linear function. Balcan, Harvey and Iwata [2] showed that every symmetric submodular

6:4 Approx Representation of Sym Submodular Functions

function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ is \sqrt{n} -approximated by a function $g : 2^V \rightarrow \mathbb{R}_{\geq 0}$ of the form $g(S) := \sqrt{\chi(S)^T M \chi(S)}$ for all $S \subseteq V$, where $\chi(S) \in \{0, 1\}^V$ is the indicator vector of S and M is a symmetric positive definite matrix. In particular, if M is a diagonal matrix, then the function g is the square root of a linear function, i.e., a concave linear function. Given the significant role of concave linear functions, we consider the hypergraph-approximability of such functions.

► **Theorem 4.** *Symmetrized concave linear functions are 128-hypergraph-approximable.*

As a special case of Theorem 4, we obtain that the symmetrized rank function of uniform matroids is constant-hypergraph-approximable. Thus, symmetrized rank functions of uniform matroids act as a starting point for identifying subfamilies of symmetrized matroid rank functions that are constant-hypergraph-approximable. We consider a generalization of the uniform matroid, namely the partition matroid and show that it is also constant-hypergraph-approximable. We refer the reader to Section 1.2 for formal definitions of uniform and partition matroids.

► **Theorem 5.** *Symmetrized rank functions of uniform matroids and partition matroids are 64-hypergraph-approximable.*

Theorem 5 gives a concrete class of functions for which there is a large gap between the approximation capabilities of graph cut functions and hypergraph cut functions. Consider the uniform matroid where the independent sets are those of size at most 1. The symmetrized rank function of this matroid is the same as the cut function of a hypergraph with a single hyperedge spanning all vertices. As mentioned above, this function cannot be $(n/4 - \epsilon)$ -approximated by a graph cut function for all constant $\epsilon > 0$ [5]. Thus, symmetrized rank functions of uniform and partition matroids cannot be better than $n/4$ approximated by graph cut functions, but can be constant factor approximated by hypergraph cut functions.

While our lower bound result in Theorem 3 rules out α -hypergraph-approximability for symmetric submodular functions for $\alpha = o(n^{1/3}/\log^2 n)$, our positive results suggest broad families of symmetric submodular functions which are constant-hypergraph-approximable. It would be interesting to characterize the family of symmetric submodular functions that are constant-hypergraph-approximable. We also do not know if our lower bound result in Theorem 3 is tight. We only know that every symmetric submodular function is $(n - 1)$ -graph-approximable. It would be interesting to show that every symmetric submodular function is $\tilde{O}(n^{1/3})$ -hypergraph-approximable – we believe that Proposition 2 and Theorem 4 should help towards achieving this approximation factor.

1.2 Preliminaries

A set function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ is submodular if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$, symmetric if $f(A) = f(V - A)$ for all subsets $A \subseteq V$, and monotone if $f(B) \geq f(A)$ for all subsets $A \subseteq B \subseteq V$.

A matroid $\mathcal{M} = (V, \mathcal{I})$ is specified by a ground set V and a collection $\mathcal{I} \subseteq 2^V$, known as independent sets, satisfying the three independent set axioms: (1) $\emptyset \in \mathcal{I}$, (2) if $B \in \mathcal{I}$, then $A \in \mathcal{I}$ for every $A \subseteq B$, and (3) if $A, B \in \mathcal{I}$ with $|B| > |A|$, then there exists an element $v \in B \setminus A$ such that $A \cup \{v\} \in \mathcal{I}$. The rank function $r : 2^V \rightarrow \mathbb{Z}_{\geq 0}$ of a matroid $\mathcal{M} = (V, \mathcal{I})$ is defined as

$$r(A) := \max\{|S| : S \subseteq A, S \in \mathcal{I}\} \quad \forall A \subseteq V.$$

The definition of matroid rank functions that we presented in Section 1.1 is equivalent to this definition [10]. It is well-known that the rank function of a matroid is monotone submodular.

We consider two matroids over the ground set V . A uniform matroid is a matroid in which the independent sets are exactly the sets containing at most k elements of the ground set V , for some fixed integer k – we call it as the uniform matroid over ground set V with budget k . Partition matroids generalize uniform matroids: the independent sets of the partition matroid associated with a partition P_1, \dots, P_t of the ground set V with budgets $b_1, \dots, b_t \in \mathbb{Z}_{\geq 0}$ are those subsets $A \subseteq V$ for which $|A \cap P_i| \leq b_i$ for every $i \in [t]$.

The proof of our lower bound will use the following theorem showing the existence of cut-sparsifiers.

► **Theorem 6** ([4]). *For every positive constant ϵ and for every weighted n -vertex hypergraph H , there exists another weighted hypergraph H' (called a cut-sparsifier) on the same vertex set with $\tilde{O}(n/\epsilon^2)$ hyperedges such that the cut function of H' $(1 + \epsilon)$ -approximates the cut function of H .*

1.3 Proof of Proposition 2

In this section, we prove Proposition 2. We need the notion of contraction of set functions and hypergraphs. For a set function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ and a subset A , the function $g : 2^{V-A+a} \rightarrow \mathbb{R}_{\geq 0}$ obtained by contracting f with respect to A is defined as

$$g(S) := \begin{cases} f(S) & \text{if } a \notin S, \\ f(S - a + A) & \text{if } a \in S. \end{cases}$$

If $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ is a symmetric submodular function and $A \subseteq V$, then the function obtained by contracting f with respect to A is also symmetric submodular. Let $H = (V, E)$ be a hypergraph with hyperedge weights $w : E \rightarrow \mathbb{R}_{\geq 0}$ and $A \subseteq V$. Then, the hypergraph obtained by contracting H with respect to A is defined as $H' = (V - A + a, E')$ where a is a new vertex not present in V and

$$E' := \{e - A + a : e \in E, e \cap A \neq \emptyset\} \cup \{e : e \in E, e \cap A = \emptyset\}.$$

We note that E' could have self-loops and that there is a surjection $\phi : E \rightarrow E'$ mapping each hyperedge to the hyperedge it is contracted into (which could be the same as the original hyperedge). We use this surjection to define the weight $w' : E' \rightarrow \mathbb{R}_{\geq 0}$ of hyperedges in E' as $w'(e') = \sum_{e \in E: \phi(e)=e'} w(e)$. We note that if f is the cut function of a weighted hypergraph $H = (V, E)$ with hyperedge weights $w : E \rightarrow \mathbb{R}_{\geq 0}$ and $A \subseteq V$, then the contraction of f with respect to A corresponds to the cut function of the weighted hypergraph (H', w') obtained by contracting H with respect to A . This leads to the following observation:

► **Observation 7.** *The contraction of a α -hypergraph-approximable function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ with respect to a subset $A \subseteq V$ is also α -hypergraph-approximable.*

► **Proposition 2.** *If the symmetrization of every matroid rank function is α -hypergraph-approximable, then every rational-valued symmetric submodular function is α -hypergraph-approximable.*

Proof. It suffices to consider integer-valued symmetric submodular functions (multiply all function values by the product of the denominators of their rational expressions). Hence, we will focus on approximating integer-valued symmetric submodular functions.

Let $f : 2^V \rightarrow \mathbb{Z}_{\geq 0}$ be an integer-valued symmetric submodular function. It is known that there exists a vector $w \in \mathbb{R}^V$ such that the function $g : 2^V \rightarrow \mathbb{R}_{\geq 0}$ defined by

$$g(S) := f(S) + \sum_{u \in S} w_u \quad \forall S \subseteq V$$

6:6 Approx Representation of Sym Submodular Functions

is integer-valued, monotone, and submodular [7, Section 3.3] (e.g., for our purposes, we can simply choose $w_u := \max\{f(S) : S \subseteq V\}$ for every $u \in V$). Since $f(V) = 0$, we have that $g(V) = \sum_{u \in V} w_u$. Consequently, $(1/2)g_{\text{sym}}(S) = (1/2)(g(S) + g(V - S) - g(V)) = (1/2)(f(S) + f(V - S)) = f(S)$ for every $S \subseteq V$ since f is symmetric. Thus, $f(S) = (1/2)g_{\text{sym}}(S)$ for every $S \subseteq V$.

Next, consider the integer-valued monotone submodular function $g : 2^V \rightarrow \mathbb{Z}_{\geq 0}$ obtained as above. Helgason [9] showed that there exists a matroid on a ground set U with rank function $r : 2^U \rightarrow \mathbb{Z}_{\geq 0}$ and a partition $(U_v : v \in V)$ of U such that $g(S) = r(\cup_{v \in S} U_v)$ for every $S \subseteq V$. Equivalently, the function g is obtained from the rank function r by repeatedly contracting with respect to U_v for each $v \in V$ (the order of processing $v \in V$ is irrelevant). Moreover, $f(S) = (1/2)g_{\text{sym}}(S) = (1/2)r_{\text{sym}}(\cup_{v \in S} U_v)$ for every $S \subseteq V$. Hence, the function f is half times the contraction of a symmetrized matroid rank function. Thus, if every symmetrized matroid rank function is α -hypergraph-approximable, then by Observation 7, the function f is also α -hypergraph-approximable. ◀

2 Lower Bound

In this section, we prove Theorem 3. In our first lemma, we show that it suffices to consider only hypergraphs with $\tilde{O}(n)$ hyperedges if we are willing to tolerate a constant loss in the approximation factor.

► **Lemma 8** (Few hyperedges suffice). *Let $f : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ be a symmetric submodular function and $\beta \geq 1$ be a positive real number. Suppose that there exists a weighted hypergraph H whose cut function β -approximates f . Then, there exists a weighted hypergraph H' with $\tilde{O}(n)$ hyperedges whose cut function 2β -approximates f .*

Proof. Applying Theorem 6 to H with $\epsilon = 1$ gives us that there exists a weighted hypergraph H' with $\tilde{O}(n)$ hyperedges whose cut function 2-approximates the cut function of H . Since the cut function of H β -approximates f , this means that the cut function of H' 2β -approximates f . ◀

Next we show that it suffices to restrict our attention to hypergraphs with rational hyperedge weights while again losing only a constant in the approximation factor (since we will be considering only hypergraphs with $O(n)$ hyperedges).

► **Lemma 9** (Bounded rational weights suffice). *Let $r : 2^V \rightarrow \mathbb{R}_+$ be a matroid rank function on ground set $V = [n]$. Suppose that there exists a hypergraph $H = (V, E)$ with hyperedge weights $w : E \rightarrow \mathbb{R}_+$ with $|E| = \tilde{O}(n)$ whose cut function $d : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ β -approximates r_{sym} for some $\beta = o(n)$. Then, there exist hyperedge weights $w' : E \rightarrow \mathbb{Q}_+$ which assign to each hyperedge of H a positive rational weight p/q where $p, q \leq n^3$ such that d' 2β -approximates r_{sym} , where $d' : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ is the cut function induced by the weight function w' .*

Proof. Since r is the rank function of a matroid on ground set $V = [n]$, we have that $r(S) \leq n$ for all $S \subseteq [n]$, and therefore $r_{\text{sym}}(S) \leq n$ for all $S \subseteq [n]$. Consequently, if $w(e) > n$ for some $e \in E$ then we have $d(S) > n \geq r_{\text{sym}}(S)$ for some $S \subseteq [n]$, a contradiction. Thus, we conclude that $w(e) \leq n$ for every $e \in E$.

We define the new weight function $w' : E \rightarrow \mathbb{R}_{\geq 0}$ by

$$w'(e) := \frac{\lfloor n^2 w(e) \rfloor}{n^2} \quad \forall e \in E.$$

For every $e \in E$, the weight $w'(e)$ is a rational number p/q with $q = n^2$ and $p \leq n^2 w(e) \leq n^3$. Next, we show that the cut function $d' : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ induced by this weight function w' satisfies the required bounds for every subset $S \subseteq [n]$.

For every $e \in E$, we have that $w'(e) \leq w(e)$. Thus, for every $S \subseteq [n]$, we have that $d'(S) \leq d(S) \leq r_{\text{sym}}(S)$. Moreover, for every $e \in E$, we have that $w'(e) \geq w(e) - 1/n^2$. Therefore, for every $S \subseteq [n]$, we have that

$$d'(S) \geq d(S) - |E|/n^2. \tag{1}$$

Let $S \subseteq [n]$. If $r_{\text{sym}}(S) = 0$, then $d'(S) \leq d(S) = 0$, and so $r_{\text{sym}}(S) \leq 2\beta d'(S)$. Suppose $r_{\text{sym}}(S) > 0$. Since $r_{\text{sym}}(S)$ is an integer, this means that $r_{\text{sym}}(S) \geq 1$, and therefore $1/\beta \leq r_{\text{sym}}(S)/\beta \leq d(S)$. Since $|E| = \tilde{O}(n)$, we have that $|E|/n^2 = \tilde{O}(1/n)$, and since $\beta = o(n)$, we conclude that $|E|/n^2 < 1/2\beta \leq d(S)/2$. Hence, Inequality (1) gives us that $d'(S) \geq d(S)/2$, and therefore, $r_{\text{sym}}(S) \leq \beta d(S) \leq 2\beta d'(S)$. ◀

We will show the existence of our desired matroid rank function using the following theorem of Balcan and Harvey [3].

► **Theorem 10** ([3]). *For every positive integer n and $k \geq 8$ with $k = 2^{o(n^{1/3})}$, there exists a family of sets $\mathcal{A} \subseteq 2^{[n]}$ and a family of matroids $\mathcal{M} = \{M_{\mathcal{B}} : \mathcal{B} \subseteq \mathcal{A}\}$ on the ground set $[n]$ with the following properties:*

1. $|\mathcal{A}| = k$ and $|A| = \lfloor n^{1/3} \rfloor$ for every $A \in \mathcal{A}$.
2. For every $\mathcal{B} \subseteq \mathcal{A}$ and every $A \in \mathcal{A}$, we have

$$\text{rank}_{M_{\mathcal{B}}}(A) = \begin{cases} 8 \lfloor \log k \rfloor & (\text{if } A \in \mathcal{B}) \\ |A| = \lfloor n^{1/3} \rfloor & (\text{if } A \in \mathcal{A} \setminus \mathcal{B}) \end{cases}.$$

3. For every $A_1, A_2 \in \mathcal{A}$ with $A_1 \neq A_2$, we have $|A_1 \cap A_2| \leq 4 \log k$.
4. For every $\mathcal{B} \subsetneq \mathcal{A}$, we have $\text{rank}_{M_{\mathcal{B}}}([n]) = \lfloor n^{1/3} \rfloor$.

We note that the version of the theorem given in [3] does not include the third and fourth properties. However, the proof for the variant of Theorem 10 with the first two properties given in [3] shows that the third and fourth properties also hold. We are now ready to prove Theorem 3. The following is a restatement of Theorem 3.

► **Theorem 11.** *For every sufficiently large positive integer n , there exists a symmetrized matroid rank function $r_{\text{sym}} : 2^{[n]} \rightarrow \mathbb{Z}_{\geq 0}$ on ground set $[n]$ such that r_{sym} is not α -hypergraph-approximable for $\alpha = o(n^{1/3}/\log^2 n)$.*

Proof. For simplicity, we will assume that $n = 8^x$ for some positive integer x , so that $\log n$ and $n^{1/3}$ are both integers. If the theorem holds for n of this form, it holds for all sufficiently large n , since for any $8^x \leq n < 8^{x+1}$ we can extend a matroid M on ground set $[8^x]$ to a matroid M' on ground set $[n]$ which has the same independent sets.

For $k = n^{\log n}$, let \mathcal{A} be a collection of subsets of $[n]$ and $\mathcal{M} = \{M_{\mathcal{B}} : \mathcal{B} \subseteq \mathcal{A}\}$ be the family of matroids on ground set $[n]$ with the properties guaranteed by Theorem 10. We note that $|\mathcal{M}| = 2^{n^{\log n}}$. For each $\mathcal{B} \subseteq \mathcal{A}$, let $r_{\text{sym}}^{\mathcal{B}}$ be the symmetrized rank function of $M_{\mathcal{B}}$ and let $\mathcal{F} := \{r_{\text{sym}}^{\mathcal{B}} : M_{\mathcal{B}} \in \mathcal{M}\}$ be the family of symmetrized rank functions of matroids in the family \mathcal{M} . We note that \mathcal{F} is a family of $2^{n^{\log n}}$ symmetrized matroid rank functions over the ground set $[n]$. We will prove that there exists $r_{\text{sym}}^{\mathcal{B}} \in \mathcal{F}$ which is not α -hypergraph-approximable. Suppose for contradiction that for every function $r_{\text{sym}}^{\mathcal{B}} \in \mathcal{F}$ there exists a hypergraph $H_{\mathcal{B}}$ such that the cut function $d_{\mathcal{B}}$ of $H_{\mathcal{B}}$ satisfies $d_{\mathcal{B}}(S) \leq r_{\text{sym}}^{\mathcal{B}}(S) \leq \alpha(n)d_{\mathcal{B}}(S)$ for all $S \subseteq [n]$.

6:8 Approx Representation of Sym Submodular Functions

Let $r_{\text{sym}}^{\mathcal{B}} \in \mathcal{F}$. By Lemma 8, there exists a weighted hypergraph H'_B with $\tilde{O}(n)$ hyperedges such that its cut function $d' : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ satisfies

$$d'(S) \leq r_{\text{sym}}^{\mathcal{B}}(S) \leq 2\alpha d'(S) \quad \forall S \subseteq [n].$$

Applying Lemma 9 to the rank function $\text{rank}_{\mathcal{B}} : 2^{[n]} \rightarrow \mathbb{Z}_{\geq 0}$ of the matroid $M_{\mathcal{B}}$ and the hypergraph H'_B gives a hypergraph H''_B with $\tilde{O}(n)$ hyperedges all of whose weights are rational values p/q with $p, q \leq n^3$ such that the cut function $d'' : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ of H''_B satisfies

$$d''(S) \leq r_{\text{sym}}^{\mathcal{B}}(S) \leq 4\alpha d''(S) \quad \forall S \subseteq [n].$$

Let \mathcal{H} be the family of weighted hypergraphs $\{H''_B : r_{\text{sym}}^{\mathcal{B}} \in \mathcal{F}\}$.

We now count the number of weighted hypergraphs in \mathcal{H} . Each hypergraph in \mathcal{H} has $\tilde{O}(n)$ hyperedges with each hyperedge having rational weight p/q where $p, q \leq n^3$. The number of potential hyperedges in a n -vertex hypergraph is $2^n - 1$, so for every $m \in \mathbb{Z}_+$ the number of simple n -vertex hypergraphs with m hyperedges is $\binom{2^n - 1}{m} = O(2^{nm})$. Consequently, the number of possible simple hypergraphs with $\tilde{O}(n)$ hyperedges is $2^{\tilde{O}(n^2)}$. The number of positive rational numbers p/q with $p, q \in [n^3]$ is at most n^6 , so the number of ways to assign a weight of this kind to each hyperedge of a hypergraph with $\tilde{O}(n)$ hyperedges is $n^{\tilde{O}(n)}$. Therefore the number of hypergraphs with $\tilde{O}(n)$ hyperedges each of which has a positive rational weight p/q where $p, q \in [n^3]$ is $2^{\tilde{O}(n^2)} n^{\tilde{O}(n)} = 2^{\tilde{O}(n^2)} = 2^{o(n \log n)}$. Hence, $|\mathcal{H}| = 2^{o(n \log n)}$.

Let $\mathcal{F}' := \{r_{\text{sym}}^{\mathcal{B}} \in \mathcal{F} : |\mathcal{B}| \leq |\mathcal{A}| - 2\}$. Since $|\mathcal{F}| = 2^{n \log n}$ and $|\mathcal{A}| = n \log n$, we have that $|\mathcal{F}'| = \Omega(2^{n \log n})$. Since $|\mathcal{F}'| = \Omega(2^{n \log n})$ while $|\mathcal{H}| = 2^{o(n \log n)}$, there must exist two distinct functions $r_{\text{sym}}^{\mathcal{B}_1}, r_{\text{sym}}^{\mathcal{B}_2} \in \mathcal{F}'$ such that there is a single weighted hypergraph $H \in \mathcal{H}$ whose cut function $d : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$ satisfies

$$d(S) \leq r_{\text{sym}}^{\mathcal{B}_1}(S) \leq 8\alpha d(S) \quad \forall S \subseteq [n] \quad \text{and} \quad (2)$$

$$d(S) \leq r_{\text{sym}}^{\mathcal{B}_2}(S) \leq 8\alpha d(S) \quad \forall S \subseteq [n]. \quad (3)$$

Since $\mathcal{B}_1 \neq \mathcal{B}_2$, at least one of $\mathcal{B}_1 \setminus \mathcal{B}_2$ and $\mathcal{B}_2 \setminus \mathcal{B}_1$ must be non-empty. We assume without loss of generality that $\mathcal{B}_1 \setminus \mathcal{B}_2 \neq \emptyset$. Let $S \in \mathcal{B}_1 \setminus \mathcal{B}_2$. By Theorem 10, we have that $\text{rank}_{M_{\mathcal{B}_1}}(S) = 8 \log^2 n$ and $\text{rank}_{M_{\mathcal{B}_2}}(S) = n^{1/3}$. Since $r_{\text{sym}}^{\mathcal{B}_1}, r_{\text{sym}}^{\mathcal{B}_2} \in \mathcal{F}'$, we have that $|\mathcal{B}_1|, |\mathcal{B}_2| \leq |\mathcal{A}| - 2$, and thus $|\mathcal{B}_1 \cup \{S\}|, |\mathcal{B}_2 \cup \{S\}| \leq |\mathcal{A}| - 1$. Therefore $\mathcal{A} \setminus (\mathcal{B}_1 \cup \{S\}), \mathcal{A} \setminus (\mathcal{B}_2 \cup \{S\}) \neq \emptyset$, so there exist sets $T_1 \in \mathcal{A} \setminus (\mathcal{B}_1 \cup \{S\}), T_2 \in \mathcal{A} \setminus (\mathcal{B}_2 \cup \{S\})$. By Theorem 10, we have that $\text{rank}_{M_{\mathcal{B}_1}}(T_1), \text{rank}_{M_{\mathcal{B}_2}}(T_2) = n^{1/3}$ and $|S \cap T_1|, |S \cap T_2| \leq 4 \log^2 n$. Therefore, $\text{rank}_{M_{\mathcal{B}_1}}(T_1 \setminus S), \text{rank}_{M_{\mathcal{B}_2}}(T_2 \setminus S) \geq n^{1/3} - 4 \log^2 n$, and so $\text{rank}_{M_{\mathcal{B}_1}}([n] \setminus S), \text{rank}_{M_{\mathcal{B}_2}}([n] \setminus S) \geq n^{1/3} - 4 \log^2 n$. Furthermore, since $T_1, T_2 \subseteq [n]$ we have that $\text{rank}_{M_{\mathcal{B}_1}}([n]), \text{rank}_{M_{\mathcal{B}_2}}([n]) \geq n^{1/3}$, and so by Theorem 10, we have $\text{rank}_{M_{\mathcal{B}_1}}([n]), \text{rank}_{M_{\mathcal{B}_2}}([n]) = n^{1/3}$. Thus, we have that

$$\begin{aligned} r_{\text{sym}}^{\mathcal{B}_1}(S) &= \text{rank}_{M_{\mathcal{B}_1}}(S) + \text{rank}_{M_{\mathcal{B}_1}}([n] \setminus S) - \text{rank}_{M_{\mathcal{B}_1}}([n]) \\ &\leq 8 \log^2 n + n^{1/3} - n^{1/3} = 8 \log^2 n \quad \text{and} \\ r_{\text{sym}}^{\mathcal{B}_2}(S) &= \text{rank}_{M_{\mathcal{B}_2}}(S) + \text{rank}_{M_{\mathcal{B}_2}}([n] \setminus S) - \text{rank}_{M_{\mathcal{B}_2}}([n]) \\ &\geq n^{1/3} + (n^{1/3} - 4 \log^2 n) - n^{1/3} = n^{1/3} - 4 \log^2 n. \end{aligned}$$

Therefore, by inequalities (2) and (3), we have that $d(S) \leq r_{\text{sym}}^{\mathcal{B}_1}(S) \leq 8 \log^2 n$, and $8\alpha d(S) \geq r_{\text{sym}}^{\mathcal{B}_2}(S) \geq n^{1/3} - 4 \log^2 n$. Hence, $\alpha = \Omega(n^{1/3} / \log^2 n)$. This contradicts the assumption that $\alpha = o(n^{1/3} / \log^2 n)$. \blacktriangleleft

3 Upper Bounds

In this section, we show that certain subfamilies of symmetric submodular functions are constant-hypergraph-approximable. In particular, we show how to approximate concave linear functions and symmetrized rank functions of uniform and partition matroids using hypergraph cut functions.

3.1 Concave Functions

In this section, we prove Theorem 4. We recall that a set function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ is a concave linear function if there exist weights $w : V \rightarrow \mathbb{R}_{\geq 0}$ and an increasing concave function $h : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ such that $f(S) = h(\sum_{v \in S} w_v)$. If all weights are one, then f_{sym} is symmetric submodular and moreover, the precise value of $f(S)$ depends only on the size $|S|$ and does not depend on the precise identify of the elements in S , so we call such functions f as anonymized concave linear functions. In Section 3.1.1, we consider the special case of anonymized concave linear functions and show that these are constant-hypergraph-approximable. We extend these ideas in Section 3.1.2 to show that symmetrized concave linear functions are constant-hypergraph-approximable.

3.1.1 Anonymized Concave Linear Functions

The following lemma is useful for proving the main theorem of this section. Its proof is given in the appendix.

► **Lemma 12.** *For every integer $n \geq 2$, $r \in \{2, \dots, n\}$, and $X \subseteq [n]$ with $1 \leq |X| \leq \frac{n}{2}$, the set of hyperedges $\delta(X)$ that cross X in a complete r -uniform n -vertex hypergraph has the following size bound:*

$$\frac{1}{4} \min \left\{ \frac{|X|r}{n}, 1 \right\} \leq \frac{|\delta(X)|}{\binom{n}{r}} \leq 4 \min \left\{ \frac{|X|r}{n}, 1 \right\}.$$

The following is the main theorem of this section.

► **Theorem 13.** *Let n be a positive real number and $h : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ be a function such that h is concave on $[0, n]$ and $h(x) = h(n-x)$ for every $x \in [0, n]$. Then, the symmetric submodular function $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ over the ground set $V = [n]$ defined by $f(S) := h(|S|) \forall S \subseteq V$ is 64-hypergraph-approximable.*

Proof. To simplify our notation, we define $a_x := h(x) - h(x-1)$ for $x \in \{1, \dots, \lceil n/2 \rceil\}$. A hypergraph is uniform if all its hyperedges have the same size and a complete t -uniform hypergraph consists of all hyperedges of size t . We define H as the union of $\lceil n/2 \rceil$ different hypergraphs, $G_0, \dots, G_{\lceil n/2 \rceil}$, each of which is a uniform hypergraph over the vertex set V and each of whose hyperedges are weighted uniformly. Formally, H is the union of:

1. A complete $\lceil \frac{n}{x} \rceil$ -uniform hypergraph G_x , with a total weight of $(a_x - a_{x+1})(x/8)$ equally distributed among its hyperedges for each $x \in \{1, \dots, \lceil n/2 \rceil - 1\}$, i.e., $w(e) = (a_x - a_{x+1})(x/8) / \binom{n}{\lceil \frac{n}{x} \rceil}$ for every hyperedge $e \in E(G_x)$ (we note that $a_x - a_{x+1} \geq 0$ since h is concave).
2. A complete 2-uniform hypergraph $G_{\lceil n/2 \rceil}$, with a total weight of $a_{\lceil n/2 \rceil}(n/32)$ equally distributed among its hyperedges.
3. A hypergraph G_0 consisting of a single n -vertex hyperedge of weight $h(0)/64$.

6:10 Approx Representation of Sym Submodular Functions

Let d be the cut function of the hypergraph H we have just defined. In order to show that d 64-approximates f , we will consider an arbitrary subset C of size k and bound its cut value in H . Since we know that d and f are both symmetric, we assume without loss of generality that $1 \leq k \leq n/2$.

We now compute the weight of hyperedges crossing C in H . We recall that $|C| = k \leq n/2$. We begin with the easy cases. $\delta(C)$ will certainly cut the single hyperedge of G_0 for a weight of exactly $h(0)/64$. The hyperedges in $G_{\lceil n/2 \rceil}$ have rank 2. Therefore, by Lemma 12, the number of hyperedges crossing C in $G_{\lceil n/2 \rceil}$ is at least a $\frac{k}{2n}$ fraction and at most a $\frac{8k}{n}$ fraction of the hyperedges in $G_{\lceil n/2 \rceil}$, for a total weight between $a_{\lceil n/2 \rceil}k/64$ and $a_{\lceil n/2 \rceil}k/4$.

Next, we compute the weight of hyperedges crossing C in G_1, \dots, G_k . Let us consider G_x for a fixed $x \in \{1, \dots, k\}$. Let $r := \lceil \frac{n}{x} \rceil$. We have that $r \geq \frac{n}{x} \geq \frac{n}{k}$, so $\frac{kr}{n} \geq 1$. Therefore, by Lemma 12, the number of hyperedges crossing C in G_x is at least a quarter of the hyperedges of G_x . We also know that even if all hyperedges in G_x cross C , the weight of those hyperedges is only $(a_x - a_{x+1})(x/8)$. Therefore, the weight of hyperedges crossing C in G_x is between $(a_x - a_{x+1})(x/32)$ and $(a_x - a_{x+1})(x/8)$.

Next, we compute the weight of hyperedges crossing C in $G_{k+1}, \dots, G_{\lceil n/2 \rceil - 1}$. Let us consider G_x for a fixed $x \in \{k+1, \dots, \lceil \frac{n}{2} \rceil - 1\}$. Let $r := \lceil \frac{n}{x} \rceil$. Then, $2 \leq r \leq \frac{2n}{x} < \frac{2n}{k}$. Therefore, $\frac{kr}{n} \leq 2$, and hence,

$$\frac{k}{2x} \leq \frac{kr}{2n} \leq \min\left(\frac{kr}{n}, 1\right) \leq \frac{kr}{n} \leq \frac{2k}{x}.$$

From these inequalities and Lemma 12, we conclude that the number of hyperedges crossing C in G_x is at least a $\frac{k}{8x}$ fraction and at most a $\frac{8k}{x}$ fraction of hyperedges of G_x . Therefore, the weight of hyperedges crossing C in G_x is at least $(a_x - a_{x+1})k/64$ and at most $(a_x - a_{x+1})k$.

Therefore, if $d(C)$ is the weight of hyperedges crossing C in H , then

$$\frac{1}{64} \left(h(0) + a_{\lceil n/2 \rceil}k + \sum_{x=1}^k (a_x - a_{x+1})x + \sum_{x=k+1}^{\lceil n/2 \rceil - 1} (a_x - a_{x+1})k \right) \quad (4)$$

$$\leq \frac{1}{64} \left(h(0) + a_{\lceil n/2 \rceil}k + \sum_{x=1}^k 2(a_x - a_{x+1})x + \sum_{x=k+1}^{\lceil n/2 \rceil - 1} (a_x - a_{x+1})k \right) \quad (5)$$

$$\leq d(C) \quad (6)$$

$$\leq \frac{h(0)}{64} + a_{\lceil n/2 \rceil} \frac{k}{4} + \sum_{x=1}^k (a_x - a_{x+1}) \frac{x}{8} + \sum_{x=k+1}^{\lceil n/2 \rceil - 1} (a_x - a_{x+1})k \quad (7)$$

$$\leq h(0) + a_{\lceil n/2 \rceil}k + \sum_{x=1}^k (a_x - a_{x+1})x + \sum_{x=k+1}^{\lceil n/2 \rceil - 1} (a_x - a_{x+1})k. \quad (8)$$

Here, expression (8) is 64 times expression (4), so our proof is complete if we can show that expression (8) evaluates to $h(k)$ (recall that $h(k) = f(C)$). The next claim completes the proof by showing this. \blacktriangleleft

\triangleright **Claim 14.** For every $k \in \{0, 1, 2, \dots, \lceil n/2 \rceil\}$, we have that

$$h(k) = h(0) + a_{\lceil n/2 \rceil}k + \sum_{x=1}^k (a_x - a_{x+1})x + \sum_{x=k+1}^{\lceil n/2 \rceil - 1} (a_x - a_{x+1})k$$

Proof. To show this, we simplify the two summations appearing on the RHS. The second summation telescopes to yield $(a_{k+1} - a_{\lceil n/2 \rceil})k$. To simplify the first summation, we note that $\sum_{x=1}^j (a_x - a_{x+1})x = \sum_{x=1}^j (2h(x) - h(x+1) - h(x-1))x$. For every x from 1 to $j-1$, $h(x)$ is added $2x$ times and subtracted $2x$ times in this summation, so it does not contribute at all. Therefore, we conclude that $\sum_{x=1}^j (a_x - a_{x+1})x = (j+1)h(x) - jh(x+1) - h(0)$.

Using the simplifications we have derived for each of the summations on the RHS, we find that the RHS is

$$\begin{aligned} & h(0) + a_{\lceil n/2 \rceil}k + ((k+1)h(k) - kh(k+1) - h(0)) + (a_{k+1} - a_{\lceil n/2 \rceil})k \\ &= ka_{k+1} + (k+1)h(k) - kh(k+1) \\ &= k(h(k+1) - h(k)) + (k+1)h(k) - kh(k+1) \\ &= h(k). \end{aligned} \quad \triangleleft$$

3.1.2 Symmetrized Concave Linear Functions

In this section, we prove Theorem 4 – i.e., symmetrized concave linear functions are constant-hypergraph-approximable. Our approach is to first construct a hypergraph on a much larger vertex set than the ground set V , using the result of Theorem 13, and then contract subsets of the vertices of this hypergraph to obtain a hypergraph on the vertex set V with the desired property.

► **Theorem 15.** *Let V be a ground set, $w: V \rightarrow \mathbb{R}_+$, and $h: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ be an increasing concave function. Then, the symmetric submodular function $f: 2^V \rightarrow \mathbb{R}_+$ defined by*

$$f(S) := h\left(\sum_{v \in S} w(v)\right) + h\left(\sum_{v \in V \setminus S} w(v)\right) - h\left(\sum_{v \in V} w(v)\right) - h(0) \quad \forall S \subseteq V$$

is 128-hypergraph-approximable.

Proof. Let $n := |V|$. For ease of notation, we will use $w(S) := \sum_{v \in S} w(v)$ for all $S \subseteq V$. Let $g: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ be defined by $g(x) := h(x) + h(w(V) - x) - h(w(V)) - h(0)$ for all $x \geq 0$. Then $f(S) = g(w(S))$. Since h is concave, and $h(w(V) - x)$ is $h(x)$ reflected over a vertical line at $x = w(V)/2$, the function $h(w(V) - x)$ is also concave. We also note that $-h(w(V)) - h(0)$ is a constant, and constant functions are concave. Therefore, g is a sum of concave functions, and hence, g is concave as well. Since g is concave, it is also continuous. Therefore, for every $x \in \mathbb{R}_+$ such that $g(x) \neq 0$, there exists a positive real number ε_x such that for every real number y with $x - \varepsilon_x < y < x + \varepsilon_x$, we have $g(x)/\sqrt{2} \leq g(y) \leq \sqrt{2}g(x)$. Let $\varepsilon_{\min} = \min\{\varepsilon_{w(S)} : \emptyset \neq S \subset V\}$. Let $q := \lceil 2nw(V)/\varepsilon_{\min} \rceil$. We note that $w(V)/q \leq \varepsilon_{w(S)}/2n$ for every $S \subset V$.

▷ **Claim 16.** There exist positive integers p_v for each $v \in V$ such that:

1. For every $v \in V$, we have that $w(v) - \frac{\varepsilon_{\min}}{n} < \frac{p_v w(V)}{q} < w(v) + \frac{\varepsilon_{\min}}{n}$.
2. $\sum_{v \in V} p_v = q$.

Proof. By our choice of q , for every $v \in V$, we have that $w(v) - w(V)/q > w(v) - \frac{\varepsilon_{\min}}{n}$. Therefore, for each $v \in V$ we can choose a positive integer p_v such that $w(v) - \frac{\varepsilon_{\min}}{n} < p_v w(V)/q \leq w(v)$, and thus we can choose a collection of integers p_v which satisfies the first condition of the claim as well as $\sum_{v \in V} p_v \leq q$.

6:12 Approx Representation of Sym Submodular Functions

Consider a collection of positive integers p_v for each $v \in V$ which maximizes $\sum_{v \in V} p_v$ subject to satisfying the first condition of the claim and the inequality $\sum_{v \in V} p_v \leq q$. Suppose for contradiction that these integers do not satisfy the second condition of the claim. Then, $\sum_{v \in V} p_v < q$, so $\sum_{v \in V} p_v w(V)/q < w(V)$, so there must exist some $u \in V$ for which $p_u w(V)/q < w(u)$. By our choice of q , we have that

$$\frac{(p_u + 1)w(V)}{q} = \frac{p_u w(V)}{q} + \frac{w(V)}{q} < w(u) + \frac{w(V)}{q} \leq w(u) + \frac{\varepsilon_{min}}{2n} < w(u) + \frac{\varepsilon_{min}}{n}.$$

Thus, we can increase p_u by 1 while still satisfying the first condition of the claim. Also, since $\sum_{v \in V} p_v < q$, we have that $1 + \sum_{v \in V} p_v \leq q$, so we can increase p_u by 1 while maintaining that the sum of all the integers p_v is at most q . This contradicts our assumption that the integers p_v maximized $\sum_{v \in V} p_v$ subject to satisfying the first constraint of the claim and the inequality $\sum_{v \in V} p_v \leq q$. Thus, a collection of positive integers satisfying the conditions of the claim exists. \triangleleft

Choose a positive integer p_v for each $v \in V$ such that the chosen integers satisfy the conditions of Claim 16. For each $v \in V$, we create a set U_v containing p_v new vertices, and we define $U := \bigcup_{v \in V} U_v$. We note that $|U| = \sum_{v \in V} p_v = q$. We define functions $h_1: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, $f_1: [0, q] \rightarrow \mathbb{R}_+$, and $f_2: [0, q] \rightarrow \mathbb{R}_+$ by $h_1(x) = h(xw(V)/q)$, $f_1(x) = h_1(x) + h_1(q - x) - h_1(q) - h_1(0)$, and $f_2(x) = f_1(x)/\sqrt{2}$. We note that h_1 is concave since it is a rescaling of h by a constant factor, and $h_1(q - x)$ is concave, since it is h_1 reflected over the vertical line at $x = q/2$. Thus, f_1 is the sum of concave functions, and hence, f_1 is concave. Finally, f_2 is a constant multiple of a concave function, so f_2 is concave as well. Furthermore, by definition, $f_2(q - x) = f_2(x)$. Applying Theorem 13 to q and f_2 , we conclude that there exists a hypergraph H' with vertex set U whose cut function d' satisfies

$$d'(S) \leq f_1(|S|)/\sqrt{2} \leq 64d'(S) \quad \forall S \subseteq U,$$

Let H be the hypergraph obtained from H' by contracting each set U_v of vertices into a vertex $v \in V$. Let d be the cut function of H . To complete the proof, we will show that $d(S) \leq f(S) \leq 128d(S)$ for every $S \subseteq V$. We first consider the special cases of $S = \emptyset$ and $S = V$. For both these cases, we have that $f(S) = 0 = d(S)$ by definition. Next, let us consider an arbitrary non-empty set $S \subset V$. Let $U_S := \bigcup_{v \in S} U_v$ be the corresponding set of vertices in H' . We note that by construction of H , we have that $d(S) = d'(U_S)$. Therefore,

$$d(S) \leq f_1(|U_S|)/\sqrt{2} \leq 64d(S). \tag{9}$$

We note that

$$|U_S| = \sum_{v \in S} |U_v| = \sum_{v \in S} p_v.$$

Therefore, by definition,

$$\begin{aligned} f_1(|U_S|) &= f_1\left(\sum_{v \in S} p_v\right) \\ &= h_1\left(\sum_{v \in S} p_v\right) + h_1\left(q - \sum_{v \in S} p_v\right) - h_1(q) - h_1(0) \end{aligned}$$

$$\begin{aligned}
&= h\left(\sum_{v \in S} \frac{p_v w(V)}{q}\right) + h\left(w(V) - \sum_{v \in S} \frac{p_v w(V)}{q}\right) - h(w(V)) - h(0) \\
&= g\left(\sum_{v \in S} \frac{p_v w(V)}{q}\right).
\end{aligned}$$

For each $v \in S$, we have that $w(v) - \varepsilon_{\min}/n < p_v w(V)/q < w(v) + \varepsilon_{\min}/n$. We also have that $|S| \leq n$. Therefore,

$$\begin{aligned}
w(S) - \varepsilon_{w(S)} &\leq w(S) - \varepsilon_{\min} \\
&\leq w(S) - \frac{|S|\varepsilon_{\min}}{n} \\
&< \sum_{v \in S} \frac{p_v w(V)}{q} \\
&< w(S) + \frac{|S|\varepsilon_{\min}}{n} \\
&\leq w(S) + \varepsilon_{\min} \\
&\leq w(S) + \varepsilon_{w(S)}.
\end{aligned}$$

So by definition of $\varepsilon_{w(S)}$, we have that

$$\frac{f(S)}{\sqrt{2}} = \frac{g(w(S))}{\sqrt{2}} \leq g\left(\sum_{v \in S} \frac{p_v w(V)}{q}\right) \leq \sqrt{2}g(w(S)) = \sqrt{2}f(S).$$

Thus $f(S)/\sqrt{2} \leq f_1(|U_S|) \leq \sqrt{2}f(S)$, and so by inequality (9) we have that

$$d(S) \leq f_1(|U_S|)/\sqrt{2} \leq f(S) \leq \sqrt{2}f_1(|U_S|) \leq 128d(S). \quad \blacktriangleleft$$

3.2 Symmetrized Matroid Rank Functions

In this section, we prove Theorem 5 which states that symmetrized rank function of uniform and partition matroids are constant-hypergraph-approximable (see Section 1.2 for definitions of uniform and partition matroids). We begin with uniform matroids.

► **Lemma 17.** *The symmetrized rank function of a uniform matroid is 64-hypergraph-approximable.*

Proof. Let $r : 2^V \rightarrow \mathbb{R}_{\geq 0}$ be the rank function of the uniform matroid on ground set V with budget k and $r_{\text{sym}} : 2^V \rightarrow \mathbb{R}_{\geq 0}$ be the symmetrized rank function. We note that $r(S) = \min\{|S|, k\}$ for every $S \subseteq V$. If $k > |V|$, then $r_{\text{sym}}(S) = 0$ for every $S \subseteq V$ and hence, r_{sym} is 1-hypergraph-approximable using the empty hypergraph. So, we may assume that $k \leq |V|$. Then, for every $S \subseteq V$, we have that

$$\begin{aligned}
r_{\text{sym}}(S) &= r(S) + r(V \setminus S) - r(V) \\
&= \min\{|S|, k\} + \min\{|V \setminus S|, k\} - \min\{|V|, k\} \\
&= \min\{|S|, |V \setminus S|, k, |V| - k\}.
\end{aligned}$$

Let $n := |V|$ and consider the function $h : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ defined by

$$h(x) = \min\{x, n - x, k, n - k\}.$$

Then, h is concave on $[0, n]$ and $h(x) = h(n - x)$ for every $x \in [0, n]$ and $r_{\text{sym}}(S) = h(|S|)$ for every $S \subseteq V$. Therefore, by Theorem 13, we have that r_{sym} is 64-hypergraph-approximable. \blacktriangleleft

Next, we show that symmetrized rank functions of partition matroids are constant-hypergraph-approximable.

► **Theorem 18.** *The symmetrized rank function of a partition matroid is 64-hypergraph-approximable.*

Proof. Let $\mathcal{M} = (V, \mathcal{I})$ be a partition matroid on ground set V with rank function $r : 2^V \rightarrow \mathbb{Z}_{\geq 0}$ that is associated with the partition V_1, \dots, V_t of the ground set V and budgets $b_1, \dots, b_t \in \mathbb{Z}_{\geq 0}$. For $i \in [t]$, we define a function $f_i : 2^{V_i} \rightarrow \mathbb{Z}_{\geq 0}$ by $f_i(S) := r_i(S) + r_i(V_i \setminus S) - r_i(V_i)$ where r_i is the rank function of the uniform matroid on ground set V_i with budget b_i . Then, the symmetrized rank function of the partition matroid \mathcal{M} can be written as $r_{\text{sym}}(S) = \sum_{i=1}^t f_i(S \cap V_i)$. Moreover, each f_i is the symmetrized rank function of a uniform matroid. By Lemma 17, for each $i \in [t]$, there exists a weighted hypergraph G_i with cut function d_i such that

$$d_i(S) \leq f_i(S) \leq 64d_i(S) \quad \forall S \subseteq V_i.$$

Let G be the hypergraph on V formed by taking the union of the hypergraphs G_i for each $i \in [t]$. Since the vertex sets of the hypergraphs G_i are pairwise disjoint, the cut function $d : 2^V \rightarrow \mathbb{R}_{\geq 0}$ of G satisfies $d(S) = \sum_{i=1}^t d_i(S \cap V_i)$, and therefore G is a weighted hypergraph which fulfills the requirements of the theorem. ◀

4 Conclusion

In this work, we investigated the approximability of symmetric submodular functions using hypergraph cut functions. We proved that it suffices to understand the approximability of symmetrized matroid rank functions. On the upper bound side, we showed that symmetrized concave linear functions and symmetrized rank functions of uniform and partition matroids are constant-approximable using hypergraph cut functions. Our upper bounds for uniform and partition matroids raise the question of whether symmetrized rank functions of constant-depth laminar matroids are constant-approximable using hypergraph cut functions. On the lower bound side, we showed that there exist symmetrized matroid rank functions on n -element ground sets that cannot be $o(n^{1/3}/\log^2 n)$ -approximated using hypergraph cut functions, thus ruling out constant-approximability of symmetric submodular functions using hypergraph cut functions. Our results raise the natural open question of whether every symmetric submodular function on n -element ground set is $O(\sqrt{n})$ -hypergraph approximable.

Our strong lower bound also raises the question of whether we could trade off approximability against the number of vertices in the hypergraph. In particular, for every symmetric submodular function $f : 2^V \rightarrow \mathbb{R}_+$ defined over a n -element ground set V , does there exist a hypergraph over a vertex set $V' \supseteq V$ with cut function $d : 2^{V'} \rightarrow \mathbb{R}_{\geq 0}$ such that $d(A) \leq f(A) \leq \alpha d(A)$ for every $A \subseteq V$, where $\alpha = O(1)$ and $|V'| = O(2^n)$?

References

- 1 A. Badanidiyuru, S. Dobzinski, H. Fu, R. Kleinberg, N. Nisan, and T. Roughgarden. Sketching valuation functions. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete algorithms*, SODA, pages 1025–1035, 2012.
- 2 M-F. Balcan, N. Harvey, and S. Iwata. Learning symmetric non-monotone submodular functions. In *NIPS Workshop on Discrete Optimization in Machine Learning*, NIPS, 2012.
- 3 M-F. Balcan and Nicholas JA Harvey. Submodular functions: Learnability, structure, and optimization. *SIAM Journal on Computing*, 47(3):703–754, 2018.

- 4 Y. Chen, S. Khanna, and A. Nagda. Near-linear size hypergraph cut sparsifiers. In *Proceedings of the IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 61–72, 2020.
- 5 N. Devanur, S. Dughmi, R. Schwartz, A. Sharma, and M. Singh. On the Approximation of Submodular Functions. Preprint in arXiv: 1304.4948v1, 2013.
- 6 V. Feldman and J. Vondrák. Optimal Bounds on Approximation of Submodular and XOS Functions by Juntas. *SIAM Journal on Computing*, 45(3):1129–1170, 2016.
- 7 S. Fujishige. *Submodular functions and optimization*. Elsevier, 2005.
- 8 M. Goemans, N. Harvey, S. Iwata, and V. Mirrokni. Approximating submodular functions everywhere. In *Proceedings of the 20th annual ACM-SIAM Symposium on Discrete algorithms*, SODA, pages 535–544, 2009.
- 9 T. Helgason. Aspects of the theory of hypermatroids. In *Hypergraph Seminar*, pages 191–213. Springer, 1974.
- 10 A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003.
- 11 C. Seshadri and J. Vondrák. Is submodularity testable. *Algorithmica*, 69(1):1–25, 2014.
- 12 Z. Svitkina and L. Fleischer. Submodular Approximation: Sampling-based Algorithms and Lower Bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011.

A Proof of Lemma 12

We first show a few combinatorial inequalities that will be useful for our proof.

▷ **Claim 19.** For every integer $n \geq 2$, $k \in \{1, \dots, \frac{n}{2}\}$, and $r \in \{2, \dots, n - k\}$, we have that

1. $\left(1 - \frac{r}{n-k}\right)^k \leq \frac{\binom{n-k}{r}}{\binom{n}{r}} \leq \left(1 - \frac{r}{n}\right)^k$.
2. $\left(1 - \frac{k}{n-r}\right)^r \leq \frac{\binom{n-k}{r}}{\binom{n}{r}}$

Proof.

1. We note that

$$\begin{aligned} \frac{\binom{n-k}{r}}{\binom{n}{r}} &= \frac{(n-k)!/(r!(n-k-r)!)}{n!/(r!(n-r)!)} \\ &= \frac{(n-k)!}{n!} \cdot \frac{(n-r)!}{(n-k-r)!} \\ &= \prod_{i=0}^{k-1} \frac{n-r-i}{n-i}. \end{aligned}$$

We get the upper bound on $\binom{n-k}{r}/\binom{n}{r}$ by upper bounding every element of this product with $\frac{n-r}{n}$, and the lower bound by lower bounding every term of the product with $\frac{n-k-r}{n-k}$.

2. We note that

$$\begin{aligned} \frac{\binom{n-k}{r}}{\binom{n}{r}} &= \frac{(n-k)!/(r!(n-k-r)!)}{n!/(r!(n-r)!)} \\ &= \frac{(n-k)!}{(n-k-r)!} \cdot \frac{(n-r)!}{n!} \\ &= \prod_{i=0}^{r-1} \frac{n-k-i}{n-i}. \end{aligned}$$

We obtain the lower bound by lower bounding every term of the product with $\frac{n-k-r}{n-r}$. ◁

6:16 Approx Representation of Sym Submodular Functions

▷ **Claim 20.** For every integer $n \geq 2$, $k \in \{1, \dots, \frac{n}{2}\}$, and $r \in \{2, \dots, n\}$, we have that

$$\frac{\binom{k}{r}}{\binom{n}{r}} \leq \left(\frac{k}{n}\right)^r.$$

Proof. If $k < r$, the bound trivially holds, because $\binom{k}{r} = 0$. Otherwise, we have

$$\begin{aligned} \frac{\binom{k}{r}}{\binom{n}{r}} &= \frac{k!/(r!(k-r)!)}{n!/(r!(n-r)!)} \\ &= \frac{k!}{(k-r)!} \cdot \frac{(n-r)!}{n!} \\ &= \prod_{i=0}^{r-1} \frac{k-i}{n-i}. \end{aligned}$$

Upper bounding every term in the product with $\frac{k}{n}$ gives the desired bound. \triangleleft

We now restate and prove Lemma 12.

► **Lemma 12.** For every integer $n \geq 2$, $r \in \{2, \dots, n\}$, and $X \subseteq [n]$ with $1 \leq |X| \leq \frac{n}{2}$, the set of hyperedges $\delta(X)$ that cross X in a complete r -uniform n -vertex hypergraph has the following size bound:

$$\frac{1}{4} \min \left\{ \frac{|X|r}{n}, 1 \right\} \leq \frac{|\delta(X)|}{\binom{n}{r}} \leq 4 \min \left\{ \frac{|X|r}{n}, 1 \right\}.$$

Proof. Let $k := |X|$. We note that the hyperedges which cross X are exactly those which are neither fully contained in X , nor fully contained in $V \setminus X$. Thus, the number of rank r hyperedges in $\delta(X)$ is exactly $\binom{n}{r} - \binom{n-k}{r} - \binom{k}{r}$.

Suppose $r > n - k$. Then, since $k \leq n/2$, we have that $r > k$ as well, so $|\delta(X)| = \binom{n}{r} - \binom{n-k}{r} - \binom{k}{r} = \binom{n}{r}$, and so we have $\frac{|\delta(X)|}{\binom{n}{r}} = 1$. Thus, we immediately have that $\frac{1}{4} \min \left\{ \frac{|X|r}{n}, 1 \right\} \leq \frac{|\delta(X)|}{\binom{n}{r}}$. Furthermore, we have that $kr > k(n-k) = kn - k^2$, so $\frac{kr}{n} > \frac{kn-k^2}{n} = k - \frac{k^2}{n} \geq k - \frac{k}{2} = \frac{k}{2}$, so we have that $\frac{|\delta(X)|}{\binom{n}{r}} \leq 4 \min \left\{ \frac{|X|r}{n}, 1 \right\}$. Henceforth we assume $r \leq n - k$.

We case on the value of k .

■ **Case 1:** $k \geq n/r$. Then $\min \left\{ \frac{|X|r}{n}, 1 \right\} = 1$. Since $\frac{|\delta(X)|}{\binom{n}{r}}$ is the fraction of the hyperedges which are in $\delta(X)$, it is trivially upper bounded by 1, and thus by $4 \min \left\{ \frac{kr}{n}, 1 \right\}$. Therefore, it remains to show the lower bound. We have that

$$\begin{aligned} \frac{|\delta(X)|}{\binom{n}{r}} &= \frac{\binom{n}{r} - \binom{n-k}{r} - \binom{k}{r}}{\binom{n}{r}} \\ &\geq 1 - \left(1 - \frac{r}{n}\right)^k - \left(\frac{k}{n}\right)^r \\ &\geq 1 - e^{-kr/n} - \left(\frac{k}{n}\right)^r \\ &\geq 1 - \frac{1}{e} - \frac{1}{4} \\ &\geq \frac{1}{4} \\ &= 0.25 \min \left\{ \frac{kr}{n}, 1 \right\}. \end{aligned}$$

Here the second line follows from the upper bound in the first conclusion of Claim 19 and the upper bound in Claim 20, and the fourth follows from our assumptions that $n/r \leq k \leq n/2$ and $r \geq 2$.

- Case 2: $k < n/r$. Then $\min\left\{\frac{|X|r}{n}, 1\right\} = \frac{kr}{n}$. Once again, we need to show a lower bound and an upper bound. We begin with the lower bound:

$$\begin{aligned}
\frac{|\delta(X)|}{\binom{n}{r}} &= \frac{\binom{n}{r} - \binom{n-k}{r} - \binom{k}{r}}{\binom{n}{r}} \\
&\geq 1 - \left(1 - \frac{r}{n}\right)^k - \left(\frac{k}{n}\right)^r \\
&\geq 1 - e^{-kr/n} - \left(\frac{k}{n}\right)^r \\
&\geq 1 - \left(1 - \frac{kr}{2n}\right) - \left(\frac{k}{n}\right)^r \\
&= \frac{kr}{2n} - \left(\frac{k}{n}\right)^r \\
&\geq \frac{kr}{2n} - \left(\frac{kr}{2n}\right)^2 \\
&= \frac{kr}{2n} \left(1 - \frac{kr}{2n}\right) \\
&\geq \frac{kr}{4n}.
\end{aligned}$$

Here the second line follows from the upper bound in the first conclusion of Claim 19 and the upper bound in Claim 20, the fourth from the Taylor expansion of e^x , the sixth from the fact that $r \geq 2$, and the last line from the assumption that $k < n/r$.

Now we show the upper bound. Since the total number of hyperedges in the graph is $\binom{n}{r}$, we have that $|\delta(X)| \leq \binom{n}{r}$. Hence, $|\delta(X)|/\binom{n}{r} \leq 1$. It remains to show that $|\delta(X)|/\binom{n}{r} \leq 4|X|r/n = 4rk/n$. We consider 3 subcases based on the values of r and k :

- Subcase 1: $r \geq n/4$. Since $r \geq n/4$ and $|X| \geq 1$, we have that $\frac{|X|r}{n} \geq \frac{1}{4}$. Therefore

$$\frac{|\delta(X)|}{\binom{n}{r}} \leq 1 \leq 4 \frac{|X|r}{n}$$

- Subcase 2: $r < n/4$ and $k < r$. In this case, we have that $\binom{k}{r} = 0$. Therefore,

$$\begin{aligned}
\frac{|\delta(X)|}{\binom{n}{r}} &= \frac{\binom{n}{r} - \binom{n-k}{r} - \binom{k}{r}}{\binom{n}{r}} \\
&= \frac{\binom{n}{r} - \binom{n-k}{r}}{\binom{n}{r}} \\
&= 1 - \frac{\binom{n-k}{r}}{\binom{n}{r}} \\
&\leq 1 - \left(1 - \frac{r}{n-k}\right)^k \\
&\leq 1 - e^{-2rk/(n-k)} \\
&\leq 1 - \left(1 - \frac{2rk}{n-k}\right)
\end{aligned}$$

6:18 Approx Representation of Sym Submodular Functions

$$\begin{aligned}
 &= \frac{2rk}{n-k} \\
 &\leq \frac{4rk}{n}.
 \end{aligned}$$

The fourth line follows from the lower bound in the first conclusion of Claim 19. The fifth line follows from observing that $0 < k/(n-r) \leq 2/3$ (since $k \leq n/2$ and $r \leq n/4$) and $\ln(1-x) \geq -2x$ for every $x \in (0, 2/3]$. The sixth line follows from the Taylor expansion of e^x , and the last line follows from the fact that $k \leq n/2$.

- Subcase 3: $r < n/4$ and $k \geq r$. In this case we have that

$$\begin{aligned}
 \frac{|\delta(X)|}{\binom{n}{r}} &= \frac{\binom{n}{r} - \binom{n-k}{r} - \binom{k}{r}}{\binom{n}{r}} \\
 &\leq \frac{\binom{n}{r} - \binom{n-k}{r}}{\binom{n}{r}} \\
 &= 1 - \frac{\binom{n-k}{r}}{\binom{n}{r}} \\
 &\leq 1 - \left(1 - \frac{k}{n-r}\right)^r \\
 &\leq 1 - e^{-2rk/(n-r)} \\
 &\leq 1 - \left(1 - \frac{2rk}{n-r}\right) \\
 &= \frac{2rk}{n-r} \\
 &\leq \frac{2rk}{3n/4} \\
 &\leq \frac{4rk}{n}.
 \end{aligned}$$

The fourth line follows from the lower bound in the second conclusion of Claim 19. The fifth line follows from observing that $0 < k/(n-r) \leq 2/3$ (since $k \leq n/2$ and $r \leq n/4$) and $\ln(1-x) \geq -2x$ for every $x \in (0, 2/3]$. The sixth line follows from the Taylor expansion of e^x . The second to last line follows from the fact that $r < n/4$. ◀

The DAG Visit Approach for Pebbling and I/O Lower Bounds

Gianfranco Bilardi ✉

Department of Information Engineering, University of Padova, Italy

Lorenzo De Stefani¹ ✉ 

Department of Computer Science, Brown University, Providence, RI, USA

Abstract

We introduce the notion of an r -visit of a Directed Acyclic Graph DAG $G = (V, E)$, a sequence of the vertices of the DAG complying with a given rule r . A rule r specifies for each vertex $v \in V$ a family of r -enabling sets of (immediate) predecessors: before visiting v , at least one of its enabling sets must have been visited. Special cases are the $r^{(top)}$ -rule (or, topological rule), for which the only enabling set is the set of all predecessors and the $r^{(sin)}$ -rule (or, singleton rule), for which the enabling sets are the singletons containing exactly one predecessor. The r -boundary complexity of a DAG G , $b_r(G)$, is the minimum integer b such that there is an r -visit where, at each stage, for at most b of the vertices yet to be visited an enabling set has already been visited. By a reformulation of known results, it is shown that the boundary complexity of a DAG G is a lower bound to the pebbling number of the reverse DAG, G^R . Several known pebbling lower bounds can be cast in terms of the $r^{(sin)}$ -boundary complexity. The main contributions of this paper are as follows:

- An existentially tight $\mathcal{O}(\sqrt{d_{out}n})$ upper bound to the $r^{(sin)}$ -boundary complexity of any DAG of n vertices and out-degree d_{out} .
- An existentially tight $\mathcal{O}\left(\frac{d_{out}}{\log_2 d_{out}} \log_2 n\right)$ upper bound to the $r^{(top)}$ -boundary complexity of any DAG. (There are DAGs for which $r^{(top)}$ provides a tight pebbling lower bound, whereas $r^{(sin)}$ does not.)
- A visit partition technique for I/O lower bounds, which generalizes the S -partition I/O technique introduced by Hong and Kung in their classic paper “I/O complexity: The Red-Blue pebble game”. The visit partition approach yields tight I/O bounds for some DAGs for which the S -partition technique can only yield a trivial lower bound.

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

Keywords and phrases Pebbling, Directed Acyclic Graph, Pebbling number, I/O complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.7

Related Version *Full Version:* <https://arxiv.org/pdf/2210.01897.pdf>

Funding *Gianfranco Bilardi:* This work was supported in part by the Italian National Center for HPC, Big Data, and Quantum Computing; by MIUR, the Italian Ministry of Education, University and Research, under PRIN Project n. 20174LF3T8 AHeAD (Efficient Algorithms for HARnessing Networked Data); and by the University of Padova, under Project CPGA³ (Parallel and Hierarchical Computing: Architectures, Algorithms, and Applications).

1 Introduction

A *visit* of a Directed Acyclic Graph (DAG) is a sequence of all its vertices. We consider different types of visits, where a type is specified by a *visit rule* r , a prescription that a vertex v can be visited only after all the vertices in one of a given family of *enabling sets* of predecessors of v have been visited. One example is the *singleton visit rule*, $r^{(sin)}$, where

¹ Corresponding author



each vertex is enabled by each singleton containing one of its predecessors. *Breadth First Search* (BFS) and *Depth First Search* (DFS) visits are special cases of $r^{(sin)}$ -visits. Another example is the *topological visit rule*, $r^{(top)}$, where a vertex v is enabled only by the set of all its predecessors. The $r^{(top)}$ -visits are exactly the topological orderings of the DAG. Many other rules are possible; for example, the enabling sets of a vertex could be those with a majority of its predecessors.

In this work, we investigate the r -boundary complexity of DAGs. The boundary complexity of G , $b_r(G)$, is the minimum integer b such that there exists an r -visit where, at each stage, for at most b of the vertices yet to be visited an enabling set has already been visited. By a reformulation of the results of Bilardi, Pietracaprina, and D'Alberto [10], in terms of the familiar concept of visit, we show that the boundary complexity of a DAG G is a lower bound to the *pebbling number* $p(G_R)$ of its reverse DAG, *i.e.*, $p(G_R) \geq b_r(G)$. The pebbling number of a DAG provides a measure of the space required by a computation with data dependences described by that DAG, in the *pebble game* framework, introduced by Friedman [18], Paterson and Hewitt [27], Hopcroft, Paul and Valiant [21]. While a pebbling game resembles a visit where each vertex can be visited multiple times, the relation between pebbling number and boundary complexity is rather subtle, as indicated by the fact that it involves graph reversal. Several pebbling lower bounds arguments in the literature (examples are mentioned in Section 3) can indeed be recast in terms of the $r^{(sin)}$ -boundary complexity, thus achieving some unification in the derivation of these results. In this context, it is natural to explore the potential of the visit approach to yield significant pebbling lower bounds for arbitrary DAGs.

Main contributions. We begin our study with the singleton rule and show that, for any DAG G with n nodes and out-degree at most d_{out} , $b_{r^{(sin)}}(G) \leq 4\sqrt{d_{out}n}$. As a universal bound, this result cannot be improved, as shown by matching existential lower bounds. With respect to pebbling, there are DAGs such that $b_{r^{(sin)}}(G) = \Omega(p(G_R))$, for which the singleton rule provides asymptotically tight pebbling lower bounds. But there are also DAGs with very low $r^{(sin)}$ -boundary complexity, where the reverse DAG has high pebbling number. For example, Paul, Tarjan, and Celoni [28] introduced a DAG, which we will denote as PTC , of n vertices and in-degree $d_{in} = O(1)$, and proved that $p(PTC) = \Theta(\frac{n}{\log n})$. This DAG can be easily modified to yield a DAG PTC^+ with $p(PTC^+) = \Theta(\frac{n}{\log n})$ and $b_{r^{(sin)}}((PTC^+)_R) = 2$, thus exhibiting a large gap between boundary and pebbling complexity.

It is natural to wonder whether other visit rules can lead to better bounds, whereas the singleton rule does not. We have then turned our attention to the topological rule showing that for any DAG G with n nodes and out-degree at most $d_{out} \geq 2$ the boundary $b_{r^{(top)}}(G) = \frac{d_{out}-1}{\log_2 d_{out}} \log_2 n$. This bound is existentially tight. It indicates that the potential of the topological rule for pebbling lower bounds is limited. However, the topological technique is not subsumed by the singleton one, as we exhibit DAGs for which topological visits yield a tight pebbling lower bound, whereas singleton visits yield a trivial lower bound.

For an arbitrary visit rule, r , we show that $b_r(G) \leq (d_{out} - 1)\ell + 1$, where ℓ is the length of the longest paths of G . This result is also existentially tight and is consistent with the known pebbling upper bound, $p(G_R) \leq (d_{out} - 1)\ell + 1$. It remains an open question whether a tight boundary-complexity lower bound to the pebbling number can always be found by tailoring the choice of r to the DAG, or there are DAGs for which the two metrics exhibit a gap for any rule.

We also exploit visits to analyze the *I/O complexity* of a DAG G , $IO(M, G)$, pioneered by Hong and Kung in [23]. This quantity is the minimum number of accesses to the second level of a two-level memory, with the first level (*i.e.*, the cache) of size M , required to compute G .

Such computation can be modeled by a game with pebbles of two colors. Let $k(G, M)$ be the smallest integer k such that the vertices of G can be topologically partitioned into a sequence of k subsets, each with a *dominator set* and *minimum set* no larger than M . (D is a dominator of U if the vertices of U can be computed from those of D . The minimum set of U contains those vertices of U with no successor in U .) Then, $IO(G, M) \geq M(k(G, 2M) - 1)$ [23]. Dominators play a role in the red-blue game (where pebbles are initially placed on input vertices which, if unpebbled, cannot be replebbled), but not in standard pebbling (where a pebble can be placed on an input vertex at any time, hence a dominator of what is yet to be computed needs not be currently in memory). Intuitively, each segment of a computation must read a dominator set D of the vertices being computed and at least $|D| - M$ of these reads must be to the second level of the memory. It is also shown in [23] that the minimum set, say Y , of a segment of the computation must be present in memory at the end of such segment, so that at least $|Y| - M$ of its elements must have been written to the second level of the memory. In the visit perspective, the minimum set emerges as the boundary of topological visits, capturing a space requirement at various points of the computation. In addition to providing some intuition on minimum sets, this insight suggests a generalization of the partitioning technique to any type of visit. In fact, the universal upper bounds on visit boundaries mentioned above do indicate that the singleton rule has the potential to yield better lower bounds than the topological one. Following this insight, we have developed the visit partition technique. For some DAGs for which S partitions can only lead to a trivial, $\Omega(1)$, lower bound, visit partitions yield a much higher and tight lower bound.

Further related work. Since the work of Hong and Kung [23], I/O complexity has attracted considerable attention, thanks also to the increasing impact of the memory hierarchy on the performance of all computing systems, from general purpose processors, to accelerators such as GPUs, FPGAs, and Tensor engines. Their *S-partition* technique has been the foundation to lower bounds for a number of important computational problems, such as the Fast Fourier Transform [23], the definition-based matrix multiplication [3, 22, 33], sparse matrix multiplication [26], Strassen’s matrix multiplication [7] (this work also introduces the “*G-flow*” technique, based on the Grigoriev flow of functions [19], to lower bound the size of dominator sets), and various integer multiplication algorithms [8, 16]. Ballard et al. [5, 4] generalized the results on matrix multiplication of [23], by means of the approach proposed by Irony, Toledo, and Tiskin in [22] based on the Loomis-Whitney geometric theorem [24], which captures a trade-off between dominator size and minimum set size. The same papers present tight I/O complexity bounds for various linear algebra algorithms for LU/Cholesky/LDLT/QR factorization and eigenvalues and singular values computation.

Four decades after its introduction, the *S-partition* technique [23] is still the state of the art for I/O lower bounds that do hold when recomputation (the repeated evaluation of the same DAG vertex) is allowed. Savage [30] has proposed the *S-span* technique, as “a slightly weaker but simpler version of the Hong-Kung lower bound on I/O time”[31]. The *S-covering* technique [10], which merges and extends aspects from both [23] and [30], is in principle more general than the *S-partition* technique and leads to interesting resources-augmentation considerations; however, we are not aware of its application to specific DAGs.

A number of I/O lower bound techniques have been proposed and applied to specific DAG algorithms for executions *without* recomputations. These include the *edge expansion* technique of [6], the *path routing* technique of [32], and the *closed dichotomy width* technique of [11]. While the emphasis in this paper is on models with recomputation, Section 5.4 does show how the visit partition technique specializes when recomputation is not allowed.

7:4 The DAG Visit Approach for Pebbling and I/O Lower Bound

Automatic techniques to derive I/O lower bounds - with and without recomputation - have been developed, in part with the goal of automatic performance evaluation and code restructuring for improving temporal locality in programs, by Elango et al. [17], Carpenter et al. [13], Olivry et al. [25].

Paper organization. The visit framework is formulated in Section 2. The relationship between boundary complexity and pebbling number is discussed in Section 3. Section 4 presents universal upper bounds to the boundary complexity. Section 5 develops the visit partition technique for I/O lower bounds. Conclusions are offered in Section 6. Proofs and technical material not included in the main body due to space limitations are presented in the Appendix or in the full version of this work [9].

2 Visits of a DAG

A *Directed Acyclic Graph* (DAG) $G = (V, E)$ consists of a finite set of *vertices* V and of a set of directed *edges* $E \subseteq V \times V$, which form no directed cycle. We say that edge $(u, v) \in E$ is directed from u to v . We let $\text{pre}(v) = \{u \mid (u, v) \in E\}$ denote the set of predecessors of v and $\text{suc}(v) = \{v, e) \in E\}$ denote the set of its successors. The maximum in-degree (resp. out-degree) of G is defined as $d_{in} = \max_{v \in V} |\text{pre}(v)|$ (resp., $d_{out} = \max_{v \in V} |\text{suc}(v)|$). Further, we denote as $\text{des}(v)$ (resp., $\text{anc}(v)$) of v 's descendants (resp., ancestors), that is, the vertices that can be reached from (resp., can reach) v with a directed path. Given $V' \subseteq V$, we say that $G' = (V', E \cap (V' \times V'))$ is the sub-DAG of G *induced by* V' .

Let $\phi = (v_1, \dots, v_i, \dots, v_j, \dots, v_k)$ be a sequence of vertices with $|\phi| = k$. Let $\phi[i] = v_i$, for $1 \leq i \leq j \leq k$, we denote as $\phi(i..j) = (v_{i+1}, \dots, v_j)$ the infix from the i -th element excluded to the j -th included. If $j < i$, $\phi(i..j)$ is the empty sequence. Depending on the context, we sometimes interpret a sequence as the set of items appearing in the sequence.

A *visit* of a DAG $G = (V, E)$ is a sequence of all its vertices, without repetitions, complying with a *visit rule*:

► **Definition 1** (Visit rule). *A visit rule for a DAG $G = (V, E)$ is a function $r : V \rightarrow 2^{2^V}$ where $r(v) \subseteq 2^{\text{pre}(v)}$ is a non-empty family of sets of predecessors of v called enablers of v . The set of visit rules of G is denoted as $\mathcal{R}(G)$.*

Intuitively, a rule $r \in \mathcal{R}(G)$ permits a vertex v to be visited only after at least one of its enablers $Q \in r(v)$ has been entirely visited.

► **Definition 2** (r -sequence and r -visit). *Given a DAG $G = (V, E)$ and a visit rule $r \in \mathcal{R}(G)$, a sequence ψ of distinct vertices is an r -sequence of G if, for every $1 \leq i \leq |\psi|$, the prefix $\psi[1..i-1]$ includes an enabler $Q \in r(\psi[i])$. The r -sequences with $|\psi| = n$ are called r -visits and their set is denoted as $\Psi_r(G)$.*

Clearly, any prefix of an r -sequence is an r -sequence. Of particular interest are the “*topological visit rule*” defined as $r^{(top)}(v) = \{\text{pre}(v)\}$ and the “*singleton visit rule*” defined as $r^{(sin)}(v) = \{\{u\} \mid u \in \text{pre}(v)\}$ if $|\text{pre}(v)| > 0$ and $r^{(sin)}(v) = \{\emptyset\}$ otherwise.

A vertex v not contained in ψ , but enabled by some non-empty set Q included in ψ , is considered to be a “boundary” vertex.

► **Definition 3** (Boundary of an r -sequence). *Given a DAG $G = (V, E)$, $r \in \mathcal{R}(G)$, and an r -sequence ψ of G , the r -boundary of ψ is defined as the set:*

$$B_r(\psi) = \{v \in V \setminus \psi \mid \exists Q \neq \emptyset \text{ s.t. } Q \in r(v) \wedge Q \subseteq \psi\}.$$

Input vertices are never contained in the boundary of any sequence since their only enabler is the empty set.

► **Definition 4** (Boundary complexity). *The r -boundary complexity of an r -sequence ψ is defined as:*

$$b_r(\psi) = \max_{i \in \{1, \dots, |\psi|\}} |B_r(\psi[1..i])|.$$

The r -boundary complexity of G is defined as the minimum r -boundary complexity among all r -visits of G :

$$b_r(G) = \min_{\psi \in \Psi_r(G)} b_r(\psi).$$

By definition, for any $r \in \mathcal{R}(G)$, any $\psi \in \Psi_{r(\text{top})}(G)$ and, for any $i = 1, 2, \dots, n$ we have $B_{r(\text{top})}(\psi[1..i]) \subseteq B_r(\psi[1..i])$; thus, $b_{r(\text{top})}(\psi) \leq b_r(\psi)$. Similarly, if $\emptyset \in r(v)$ only when v has no predecessors, then $b_r(\psi) \leq b_{r(\text{sin})}(\psi)$, for any $\psi \in \Psi_r(G)$.

3 Boundary complexity and pebbling number

In this section, we discuss an interesting relationship between the pebbling number of a DAG $G = (V, E)$ and the boundary complexity of its *reverse* DAG $G_R = (V, E_R)$, where $E_R = \{(u, v) | (v, u) \in E\}$, which can prove useful in deriving pebbling lower bounds.

► **Theorem 5** (Pebbling lower bound). *Let G_R be the reverse of $G = (V, E)$. Then, for any $r \in \mathcal{R}(G_R)$, the pebbling number of G satisfies:*

$$p(G) \geq b_r(G_R) = \min_{\psi \in \Psi_r(G_R)} b_r(\psi).$$

In general, the analysis of the boundary complexity is simpler than the analysis of the pebbling number, in part because, in a visit, a vertex can occur only once, whereas, in a pebbling schedule, a vertex can occur any number of times.

The proof of the preceding theorem, given in Appendix, is a reformulation of a result obtained by Bilardi et al. in [10]. They introduce the *Marking Rule technique*, which is applied to DAG G rather than to its reverse. The advantage of visits over markings lies in a more direct leverage of intuition, given the widespread utilization of various kinds of visits (*e.g.*, breadth-first search, depth-first search, topological ordering) in the theory and applications of graphs.

We will explore the potential of the visit approach to yield interesting lower bounds for specific DAGs in the next section. Here, we investigate whether Theorem 5 could be strengthened by restricting the set of r -visits ψ among which $b_r(\psi)$ is minimized. The answer turns out to be negative. To clarify in what sense, we need to consider that the proof of the theorem is based on mapping each pebbling schedule π of G to an r -visit $f_r(\pi)$ of G_R , such that the boundary of each prefix of $f_r(\pi)$ is completely covered with pebbles at some stage of π . In terms of such mapping, we have:

► **Lemma 6** (Visit from pebbling schedule). *For any $r \in \mathcal{R}(G_R)$ and any $\psi \in \Psi_r(G_R)$, there exists a pebbling schedule π of G such that $f_r(\pi) = \psi$.*

Theorem 5 provides a general approach for obtaining pebbling lower bounds, which encompasses a number of arguments developed in the literature to analyze DAGs such as directed trees [27], pyramids [29] and stacks of superconcentrator [21]. A reformulation of these arguments within the visit framework can be found in [15] stacks of superconcentrators and in [9, Section 3.1] for q -pyramid and q -complete trees DAGs.

4 Upper bounds on boundary complexity

It is natural to wonder whether the pebbling lower bound of Theorem 5 is tight. As we will see in this section, both the singleton and the topological rules, while providing tight bounds for some DAGs, yield weak lower bounds for others. Whether a tight lower bound could be obtained for any DAG G , by tailoring the visit rule to G , does remain an open question.

In particular, we will establish universal upper bounds on the boundary complexity of any DAG, with respect to any rule, in terms of outdegree and depth. We will also establish (different) universal upper bounds for both the singleton and the topological rule in terms of outdegree and the number of vertices. Before presenting these results, we introduce the notion of enabled reach, a particular set of vertices associated with a vertex v , in the context of a partial visit that includes v . This concept will play a role in the derivation of each of the three universal upper bounds.

4.1 The *enabled reach* of a vertex

In the construction of a visit sequence, we will use a divide and conquer approach whereby, having constructed a prefix ψ of the sequence, the next segment, ϕ , of the sequence is obtained by visiting a suitably chosen sub-DAG, $G' = (V', E')$, according to an appropriate rule r' . It is useful for the boundary of G' to be “self-contained” in the sense that its visit does not generate any boundary outside V' . If this is the case, the boundary of $\psi\phi$ will be a subset of the boundary of ψ , so that the visit of G' contributes to the reduction of both the set of vertices yet to be visited and the current boundary. The enabled reach, a set of vertices introduced next, induces a sub-DAG G' with the desired properties.

► **Definition 7** (Enabled reach). *Let $G = (V, E)$ and $r \in \mathcal{R}(G)$. Given an r -sequence ψ and a vertex $v \in \psi$, the r -enabled reach of v given ψ is the set:*

$$\text{reach}_r(v|\psi) = \{u \mid \exists \psi'u \subseteq \text{des}(v) \text{ s.t. } \psi\psi'u \text{ is an } r\text{-sequence}\}.$$

Intuitively, we can think of the r -enabled reach of a vertex v given an r -sequence ψ as the set of all the descendants of v which can be visited by extending ψ only with descendants of v . As an example, for $r^{(\text{sin})}$, we have that the $r^{(\text{sin})}$ -enabled reach of a vertex v given a ψ corresponds to the set of the descendants of v not in ψ . The enabled reach exhibits the following crucial property:

► **Lemma 8** (Enabled Reach Property). *Given $G = (V, E)$, $v \in V$ and $r \in \mathcal{R}(G)$, let ψ be an r -sequence including v . Let $G' = (V', E')$ be the sub-DAG induced by $V' = \text{reach}_r(v|\psi)$. Let $r' \in \mathcal{R}(G')$ be such that $r'(v) = \{Q \setminus \psi \mid Q \in r(v) \wedge Q \setminus \psi \subseteq V'\}$, for all $v \in V'$. If $\psi' \in \Psi_{r'}(G')$ then (a) $\psi\psi'$ is a r -sequence of G ; (b) for any $i = 1, \dots, |\psi'|$, $B_r(\psi\psi'[1..i]) \subseteq B_r(\psi) \cup B_{r'}(\psi'[1..i])$; and (c) $b_r(\psi\psi') \leq b_r(\psi) + b_{r'}(\psi')$.*

Proof. By definition, $Q' \in r(v)$ if and only if there exists $Q \in r(v)$ such that $Q' = Q \setminus \psi$. By construction, for any $1 \leq j \leq |\psi'|$, a vertex v appears in $\psi'[1..j]$ if there exists $Q' \in r'(v)$ such that $Q' \in \psi'[1..j]$ which implies there exists $Q \in \psi\psi'[1..j]$, and, thus, $\psi\psi'[1..j]$ is an r -sequence.

Recall that a vertex appears in the boundary of a r sequence if it is enabled but not visited. By construction, $\psi' = \text{reach}(v|\psi)$, thus, by definition, any vertex which is enabled by a subset of $\psi\psi''$ must either be included in $B_r(\psi)$ or must be included among the vertices of $\text{reach}(v|\psi)$ not yet visited in $\psi\psi'[1..j]$ and enabled by ψ' , that is $B_{r'}(\psi'[1..i])$. Hence, we have that $b_r(\psi\psi') \leq b_r(\psi) + b_{r'}(\psi')$. ◀

Lemma 8 states that visiting the r -enabled reach of a vertex v given a r -sequence ψ of G does not enable any vertex outside $\text{reach}_r(v|\psi)$ which was not enabled by ψ alone. Therefore, once the sub-DAG induced by $\text{reach}_r(v|\psi)$ is visited, the only vertices left in the r -boundary are those enabled by ψ that have not been visited thus far.

The enabled reach will be a key ingredient in the construction of visits in the next three subsections. The choice of both r -sequence ψ and of vertex v has to be tailored to the particular r . Also, highly influenced by r are the size of the boundary of ψ and the reduction achieved by G' in the parameters (*e.g.*, depth or number of vertices) governing the boundary complexity, hence the shape of the resulting bound.

4.2 General rules

The *topological depth* of a DAG is the length (*i.e.*, number of edges) of its longest directed paths. The boundary complexity, according to any visit rule, can be bounded in terms of the depth and the out-degree. The basic property that is exploited is that if b is a successor of a , then the depth of the sub-DAG induced by the descendants of b is smaller than the depth of the sub-DAG induced by the descendants of a .

► **Theorem 9** (Topological-depth general boundary complexity upper bound). *Consider $G = (V, E)$ with maximum out-degree d_{out} and topological depth ℓ . For any visit rule $r \in \mathcal{R}(G)$, there exists an r -visit $\psi \in \Psi_r(G)$ such that $b_r(\psi) \leq (d_{out} - 1)\ell + 1$.*

Proof of Theorem 9 is presented in the appendix. This upper bound is *existentially tight*: For some visit rules r , there exist some DAGs for which $b_r(G) = \Theta(d_{out}\ell)$. An example is given by the reverse q -pyramid DAG, discussed in the extended version of this work [9, Section 3.1], for which $b_{r^{(sin)}}(G) = \Theta(d_{out}\ell) = \Theta(\sqrt{qn})$, since $d_{out} = q$ and $\ell = \sqrt{n/q}$.

Below, we derive universal upper bounds for the singleton and the topological rule, which are expressed in terms of n and d_{out} . These bounds are tighter than that of Theorem 9 for DAGs with ℓ suitably large (as a function of n and d_{out}).

4.3 Singleton rule $r^{(sin)}$

The $r^{(sin)}$ rule has the interesting property that the enabled reach of v , given ψ , contains all the descendants of v not in ψ . One can easily find a v with suitably few descendants, say, less than $n/2$. A $r^{(sin)}$ -sequence ψ that contains v can be obtained as the sequence of vertices on a path from an input to v . If this path has length k , then its boundary could be of a size as big as $(d_{out} - 1)k + 1$; therefore, a small k is a prerequisite to guaranteeing small boundary complexity. In general, a good enough upper bound to k cannot be guaranteed for the entire DAG. However, it is possible to partition the DAG into a sequence of “blocks” such that (i) blocks can be visited one at a time in the order they appear in the sequence; (ii) there is a reasonably small upper bound ($O(\sqrt{d_{out}n})$) on the number of nodes that are enabled by the nodes in a block, but lie outside the block, and they lie all in the next block; and (iii) in each block, each node is reachable from one of the block inputs by a path of reasonably small length ($O(\sqrt{d_{out}n})$). When the details are filled in, the outlined approach yields the following results.

► **Theorem 10.** *Given an $G = (V, E)$ with $|V| = n$ and maximum out-degree at most d_{out} there exists a visit $\psi \in \Psi_{r^{(sin)}}(G)$ s.t. $b_{r^{(sin)}}(\psi) \leq 4(\sqrt{2} + 1)\sqrt{d_{out}n}$.*

Proof. For $d_{out} = 0$, all vertices in G are isolated and, having no predecessors, are enabled only by the empty set. Thus, no vertex belongs to the $r^{(sin)}$ -boundary of any $r^{(sin)}$ -visit ψ , hence $b_{r^{(sin)}}(\psi) = 0$, and the stated bound holds. In the sequel, we assume $d_{out} \geq 1$, and proceed by induction on n .

Base: For $n = 1$, that is, $G = (\{u\}, \emptyset)$, the statement is trivially verified as v is an input vertex and the only $r^{(sin)}$ -visit, $\psi = u$, has $r^{(sin)}$ -boundary complexity zero.

Inductive step ($n \geq 2$):

Case 1: $|I| > 1$. Here, no vertex v is an ancestor of all vertices. Let $v \in I$ and let ψ' be a $r^{(sin)}$ -visit of the DAG induced by $\text{des}(v)$, with boundary complexity at most at most $c\sqrt{d_{out}n}$, which does exist by the inductive hypothesis, since $|\text{des}(v)| < n$. Similarly, let ψ'' be a $r^{(sin)}$ -visit of the DAG induced by $V \setminus \text{des}(v)$, with boundary complexity at most $c\sqrt{d_{out}n}$. Clearly, $\psi = \psi'\psi'' \in \Psi_{r^{(sin)}}(G)$. By the definition of enabled reach, for $r^{(sin)}$, we have that $\text{reach}_{r^{(sin)}}(v|v) = \text{des}(v) \setminus \{v\}$. Hence, by Lemma 8, $B_{r^{(sin)}}(\psi') = \emptyset$. The stated bound follows.

Case 2: $|I| = 1$. We partition V into non-empty “levels” $L(1), \dots, L(\ell_s)$, such that $v \in L(i)$ if and only if the shortest directed path from u to v has length i . This path is also a $r^{(sin)}$ -sequence. Further, the r -boundary of any $r^{(sin)}$ -sequence of G included in the first i levels is a subset of the first $i+1$ levels.

We say that level $L(i)$ is a *bottleneck* if $|L(i)| \leq \gamma\sqrt{d_{out}n}$ and let $i_1 < i_2 < \dots < i_k$ denote the indices of the bottlenecks. Here, $\gamma > 0$ is a constant, whose value will be determined in the course of the proof. Conventionally, we also let $i_{k+1} = \ell_s + 1$ and $L(\ell_s + 1) = \emptyset$. Since $L(1) = \{u\}$, we have that $i_1 = 1$. We group consecutive levels into *blocks* $V_j = \cup_{i_j \leq l < i_{j+1}} L(l)$, for $j = 1, 2, \dots, k$, so that the j -th block begins with the j -th bottleneck and ends just before the $(j+1)$ -st one, or with the last level, $L(\ell_s)$, if $j = k$. The number of levels of a block is upper bounded as $i_{j+1} - i_j \leq \frac{\sqrt{n}}{\gamma\sqrt{d_{out}}}$.

We construct an $r^{(sin)}$ -visit of the form $\psi = \psi_1\psi_2 \dots \psi_k$, where ψ_j is a $r^{(sin)}$ -visit of the sub-DAG G_j induced by block V_j . Since the V_j 's partition V , $\psi \in \Psi_{r^{(sin)}}(G)$. By the properties of the levels mentioned above, $B_{r^{(sin)}}(\psi_1 \dots \psi_{j-1}) = L(i_j)$. Furthermore, as V_1, \dots, V_{j-1} have already been visited by $\psi_1 \dots \psi_{j-1}$ and ψ_j is a $r^{(sin)}$ -visit of G_j , any of its prefixes ψ'_j may only enable vertices in V_j (*i.e.*, the boundary of ψ'_j in G_j) and vertices in $L(i_{j+1})$ (*i.e.*, the children of vertices in $\psi'_j \subseteq V_j$, which are not in $V_1 \cup \dots \cup V_j$). Therefore,

$$\begin{aligned} b_r^{(sin)}(\psi) &\leq \max_j \{|L(i_j)| + |L(i_{j+1})| + b_{r^{(sin)}}(\psi_j)\} \\ &\leq 2\gamma\sqrt{d_{out}n} + \max_j \{b_{r^{(sin)}}(\psi_j)\}. \end{aligned}$$

A case analysis shows how each ψ_j can be chosen so that the above term is at most $c\sqrt{d_{out}n}$.

Case 2.1. $|V_j| \leq n/2$. By the inductive hypothesis, there exists $\psi_j \in \Psi_{r^{(sin)}}(G_j)$ such that $b_{r^{(sin)}}(\psi_j) \leq c\sqrt{d_{out}n}/2 = \frac{c}{\sqrt{2}}\sqrt{d_{out}n}$. A sufficient condition for the desired result is that $2\gamma + \frac{c}{\sqrt{2}} \leq c$, which we will discuss below.

Case 2.2. $|V_j| > n/2$. Let u_1, u_2, \dots, u_d be the input vertices of G_j .

Case 2.2.a. No input of G_j has more than $n/2$ descendants, in G_j . Then, we construct an $r^{(sin)}$ -visit of G_j as $\psi_j = u_1\psi_{u_1}u_2\psi_{u_2} \dots, u_d\psi_{u_d}$, where ψ_{u_l} is a $r^{(sin)}$ -visit, with minimum boundary complexity, of the sub-DAG induced by the descendants of u_l in G_j which have

not been visited in $u_1\psi_{u_1}u_2\psi_{u_2}\dots, u_l$. This is the $r^{(sin)}$ -enabled reach of u_l in G_j , given $u_1\psi_{u_1}u_2\psi_{u_2}\dots, u_l$. Since the input vertices of G_j are not in the boundary of any prefix of ψ_j , by Lemma 8,

$$b_r^{(sin)}(\psi_j) \leq \max_{l=1,\dots,d} b_{r^{(sin)}}(\psi_{u_l}) \leq c\sqrt{d_{out} \frac{n}{2}},$$

where the last step follows by the inductive hypothesis, considering that $\psi_{u_j} \subseteq \text{des}(u_j)$ and, by the assumption of this case, $|\text{des}(u_j)| \leq n/2$ (here, and throughout the rest of this proof, $\text{des}(v)$ refers to the descendants of v in G_j). The sufficient condition for the desired result is the same as in Case 2.1

Case 2.2.b. There is an input of G_j , w.l.o.g, say u_1 , such that $|\text{des}(u_1)| > n/2$. In order to break down V_j into pieces of size smaller than $n/2$, to be visited one at a time, we select a vertex $y \in \text{des}(u_1)$ such that $|\text{des}(y)| \geq n/2$ and $\max_{z \in \text{suc}(y)} |\text{des}(z)| < n/2$. Specifically, we can choose y as the last vertex, in a topological ordering of G_j , with at least $n/2$ descendants. Let $y \in L(i)$, where clearly $i_j \leq i < i_{j+1}$, and consider a shortest path πy among those from input vertices of G_i to y . We have $|\pi| = i - i_j < i_{j+1} - i_j \leq \frac{\sqrt{n}}{\gamma\sqrt{d_{out}}}$.

Consider now the visit $\psi_j = \pi y \psi_y \psi'$ such that ψ_y is a $r^{(sin)}$ -visit of the sub-DAG induced by $V_j \cap \text{reach}_{r^{(sin)}}(y|\pi y)$ constructed as discussed in Case a, and ψ' is a $r^{(sin)}$ -visit, with minimum boundary complexity, of the sub-DAG induced by $V_j \setminus \pi y \psi_y$. The boundary associated with any prefix of π includes at most d_{out} successors for each of its vertices, which are at most $|\pi| \leq \frac{\sqrt{n}}{\gamma\sqrt{d_{out}}}$. Thus, the total contribution of π to the boundary is at most $\frac{\sqrt{d_{out}n}}{\gamma}$. Arguing along the lines of Case a, it can be shown that $b_{r^{(sin)}}(\psi_y) \leq \frac{c}{\sqrt{2}}\sqrt{d_{out}n}$. Finally, since $|\psi_{v^*}| \geq n/2$, then $|V_j \setminus \pi y \psi_y| < n/2$. Hence, by the inductive hypothesis, $b_{r^{(sin)}}(\psi') < \frac{c}{\sqrt{2}}\sqrt{d_{out}n}$.

By Lemma 8, we can conclude that $\psi_j \in \Psi_{r^{(sin)}}(G_j)$ and

$$b_{r^{(sin)}}(\psi_j) \leq \frac{1}{\gamma}\sqrt{d_{out}n} + \max\{b_{r^{(sin)}}(\psi_y), b_{r^{(sin)}}(\psi')\} \leq \left(\frac{1}{\gamma} + \frac{c}{\sqrt{2}}\right)\sqrt{d_{out}n}.$$

To establish the stated result, we need to satisfy the bound $2\gamma + \frac{1}{\gamma} + \frac{c}{\sqrt{2}} \leq c$. This requirement is more stringent than the one for Case 2.1 and Case 2.2.a. Solving for c , the sufficient condition is $c \geq \sqrt{2}(\sqrt{2} + 1)\left(2\gamma + \frac{1}{\gamma}\right)$. The r.h.s. is minimized when we let $\gamma = \frac{1}{\sqrt{2}}$, which yields $c \geq 4(\sqrt{2} + 1)$. \blacktriangleleft

The upper bound in Theorem 10 is existentially tight as there exist DAGs, such as q -pyramids discussed in [9, Section 3.1], of matching $r^{(sin)}$ -boundary complexity.

4.4 Topological rule $r^{(top)}$

The following property is peculiar to the $r^{(top)}$ -enabled reach:

► **Lemma 11** (Disjointness of enabled $r^{(top)}$ -reach). *Let ψ be a $r^{(top)}$ -sequence of DAG $G = (V, E)$ and let $u, v \in b_{r^{(top)}}(\psi)$ be distinct vertices. Then, $\text{reach}_{r^{(top)}}(u|\psi u) \cap \text{reach}_{r^{(top)}}(v|\psi v) = \emptyset$.*

The proof is presented in the Appendix. The disjointedness of the enabled-reach sets ensures that, if there are k vertices in the boundary, at least one of them has an enabled reach with fewer than n/k vertices. By leveraging this property, we obtain the following result.

► **Theorem 12** (Upper bound boundary complexity $r^{(top)}$ -visits). *For any DAG $G = (V, E)$, there exists a visit $\psi \in \Psi_{r^{(top)}}(G)$ such that (a) if $d_{out} = 0$, then $b_{r^{(top)}}(\psi) = 0$; (b) if $d_{out} = 1$, then $b_{r^{(top)}}(\psi) = 1$; and (c) if $d_{out} \leq D$, for some $D \geq 2$, then $b_{r^{(top)}}(\psi) \leq \frac{D-1}{\log_2 D} \log_2 n + 1$.*

The proof of Theorem 12 is presented in the Appendix. The upper bound in Theorem 12 is existentially tight as there exist DAGs such as inverted q -trees (discussed in the extended version of this work [9, Section3.1]) with matching $r^{(top)}$ -boundary complexity.

Interestingly, there exist DAGs for which the $r^{(top)}$ -boundary complexity is asymptotically higher than the $r^{(sin)}$ -boundary complexity. One such DAG G is shown in Figure 1. It is a simple exercise to prove that $b_r^{(sin)}(G) = 1$, $b_r^{(top)}(G) = \Theta(\log n)$, and $p(G_R) = \Theta(\log n)$.

5 Visits and I/O complexity

In this section, we show how the visit framework is fruitful in the investigation not just of space complexity but also of I/O complexity, by developing a new I/O lower bound technique, named “*visit partition*”. This extends a result by Hong and Kung [23], which has provided the basis for many I/O lower bounds in the literature thus far.

The I/O model of computation is based on a system with a memory hierarchy of two levels: a fast memory or *cache* of M memory words and a *slow memory*, with an unlimited number of words. We assume that any value associated with a DAG vertex can be stored within a single memory word. A *computation* is a sequence of steps of the following types: (i) *operations*, with operands and results in cache; (ii) *reads*, that copy the content of a memory location into a cache location; and (iii) *writes*, that copy the content of a cache location into a memory location. Reads and writes are also called I/O operations. Input values are assumed to be available in the slow memory at the beginning of the computation. Output values are required to be in the slow memory at the end of the computation. The conditions under which a computation is a valid execution of a given DAG are intuitively clear. They can be formalized in terms of the “*red-blue pebble game*”, introduced by Hong and Kung in [23]. The *I/O complexity* $IO(G, M)$ of a DAG G is defined as the minimum number of I/O operations over all possible computations of G . The *I/O write complexity* $IO_W(G, M)$ and the *I/O read complexity* $IO_R(G, M)$ are similarly defined.

The visit partition approach to I/O lower bounds develops along the following lines:

- A visit rule r is chosen for the analysis.
- A procedure is specified to map each computation ϕ of the given DAG G to an r -visit ψ of the reverse DAG G^R .
- Given any partition of visit ψ into consecutive segments, a set of I/O operations is identified for each segment, the sets of different segments being disjoint, whence their contributions can be added in the I/O lower bound.
- The number of read operations associated with a segment is lower bounded in terms of the size of a minimum post-dominator of the segment.
- The number of write operations associated with a segment is lower bounded in terms of the number of segment vertices that either inputs of G_R (hence, outputs of G), or belong to the boundary of the visit at the beginning of the segment. The resulting global lower bound, modified by the addition of the term $|I| - |O|$, also applies to read operations.
- A lower bound, q , to the I/O of a given DAG, can be established by showing that, *for each visit, there exists* a segment partition that requires at least q I/O operations, between reads and writes.

The technical details of this outline are presented in the next subsections.

5.1 Segment partitions of a visit

The concept of *post-dominator set* mirrors that of *dominator set* used in [23]:

► **Definition 13** (Post-dominator set). *Given $H = (W, F)$ and $X \subseteq W$, a set $P \subseteq W$ is a post-dominator set of $V' \subseteq W$ if every directed path from a vertex in V' to an output vertex intersects P . We denote as $pd_{min}(X)$ the minimum size of any post-dominator set of X .*

It is simple to see that P is a post-dominator set of X in H if and only if P is a dominator set of X in the reverse DAG H_R .

Let ψ be an r -visit of $G_R = (V, E_R)$. A *segment partition* of ψ into k segments is identified by a sequence of indices $\mathbf{i} = (i_1, i_2, \dots, i_k)$, with $1 \leq i_1 < i_2 < \dots < i_k = n$. We also let $i_0 = 0$, for convenience. Since ψ is a permutation of the vertices in V , the segments partition V . For $1 \leq j \leq k$, $\psi(i_{j-1}..i_j]$ is called the j -th *segment* of the partition. Two measures play a role in our I/O lower bound analysis of any segment:

- The size, $pd_{min}(\psi(i_{j-1}..i_j])$, of minimum post-dominator sets of $\psi(i_{j-1}..i_j]$.
- The r -entering boundary size

$$b_r^{(ent)}(\psi(i_{j-1}..i_j]) = |B_r^{(ent)}(\psi(i_{j-1}..i_j])|$$

where, denoting as I_R the set of input vertices of G_R ,

$$B_r^{(ent)}(\psi(i_{j-1}..i_j]) = (I_R \cup B_r(\psi[1..i_{j-1}])) \cap \psi(i_{j-1}..i_j].$$

5.2 Lower bound

Let $\phi = (\phi_1, \phi_2, \dots, \phi_T)$ be a computation of a DAG $G = (V, E)$ in the I/O model, in T steps, where ϕ_t is the t -th step. Given an $r \in \mathcal{R}(G_R)$, we construct an r -visit ψ of G_R corresponding to ϕ . Our I/O lower bounds will be based solely on properties of the visit. To construct ψ , the computation is examined backward, one step at a time. The visit is constructed incrementally, by extending an initially empty prefix. Let $\psi[1..i(t)]$ be the prefix already constructed just before processing computation step ϕ_t (initially, $i(T) = 0$). For $t = T, T-1, \dots, 1$, if ϕ_t is either a functional operation evaluating vertex $v \in V \setminus I$ or a read operation copying a vertex $v \in I$ (input of G) into the cache, then, if v has not already been visited (*i.e.*, $v \notin \psi[1..i(t)]$) and at least one enabler set $Q \in r(v)$ has already been visited (*i.e.*, $Q \subseteq \psi[1..i(t)]$), then v is added to the visit (*i.e.*, $i(t-1) = i(t)+1$ and $\psi[1..i(t-1)] = \psi[1..i(t)] \cup v$). Otherwise, the visit constructed thus far remains unchanged (*i.e.*, $i(t-1) = i(t)$).

By construction, a vertex is included in ψ at most once and only after at least one of its enablers has been visited. To conclude that ψ is indeed an r -visit of G_R it remains to show that it contains all the vertices. The vertices in I_R (which are the inputs of G_R , that is, the outputs of G) are added to the visit when they are first encountered in the backward processing of ϕ , since they are enabled by the empty set. Suppose now, by contradiction, that there are vertices in $V \setminus I_R$ that are not included in ψ . Then, let t_v be the smallest t such that v is computed (if $v \in V \setminus I$) or read from slow memory (if $v \in I$) in step ϕ_t , and let u be the vertex with the largest t_u that is not in ψ . It must be the case that if $\psi[1..i(t_u)]$ does not include any enabler of u , which in turn implies that there exists a predecessor w of u (in G_R) that does not belong to $\psi[1..i(t_u)]$. Since $t_w > t_u$, this implies that $\psi[1..i(t_w)]$ does not include any enabler of w , whence $w \notin \psi$, which contradicts the definition of u .

7:12 The DAG Visit Approach for Pebbling and I/O Lower Bound

The procedure just described to construct an r -visit of G_R from an I/O computation ϕ of G is quite similar to the one in the proof of Theorem 5 for the analysis of the pebbling number. The differences are due to the circumstance that the standard I/O model assumes the inputs to be initially available in slow memory, whereas the pebbling model assumes that the inputs can be (repeatedly) loaded into the working space at any time.

► **Lemma 14** (Visit partition). *Let ϕ be a computation of DAG $G = (V, E)$ on the I/O model with a cache of M words. Let $r \in \mathcal{R}(G_R)$ and let ψ be the r -visit of G_R constructed from ϕ , as described above, and $\mathbf{i} = (i_1, i_2, \dots, i_k)$ any of its segment partitions. Then, the number $IO_{\mathcal{W}}(\phi, M)$ of write I/O operations and the number $IO_{\mathcal{R}}(\phi, M)$ of read I/O operations executed by ϕ satisfy the bounds*

$$IO_{\mathcal{W}}(\phi, M) \geq \mathcal{W}_r(\mathbf{i}, \psi, M) := \sum_{j=1}^k \max\{0, b_r^{(ent)}(\psi(i_{j-1}..i_j)) - M\}, \quad (1)$$

$$IO_{\mathcal{R}}(\phi, M) \geq \mathcal{W}_r(\mathbf{i}, \psi, M) + |I| - |O|, \quad (2)$$

$$IO_{\mathcal{R}}(\phi, M) \geq \mathcal{R}_r(\mathbf{i}, \psi, M) := \sum_{j=1}^k \max\{0, pd_{min}(\psi(i_{j-1}..i_j)) - M\}. \quad (3)$$

The total number $IO(\phi, M) = IO_{\mathcal{W}}(\phi, M) + IO_{\mathcal{R}}(\phi, M)$ of I/O operations executed by ϕ satisfies the bound

$$IO(\phi, M) \geq \mathcal{W}_\psi(\mathbf{i}, M) + \max\{\mathcal{R}_\psi(\mathbf{i}, M), \mathcal{W}_\psi(\mathbf{i}, M) + |I| - |O|\}. \quad (4)$$

Proof. We will analyze the entering boundary and the post-dominator contributions to the lower bound on the number of, respectively, write and read I/O operations for a generic segment of the visit $\psi(h..i)$, with $0 \leq h < i \leq n$, and then compose the contributions of the segments in partition \mathbf{i} . For $l = 1, \dots, n$, we let τ_l be such that $\psi[l]$ has been added to visit ψ in correspondence of computation step ϕ_{τ_l} . Observe that $\tau_1 > \tau_2 > \dots > \tau_n$, since the visit is constructed from the computation in reverse. When speaking of cache or of the slow memory at time t , we refer to their state just before the execution of computation step ϕ_t .

Proof of (1) – Boundary bound. We claim that, at time τ_h , the value of each vertex of set $B_r^{(ent)}(\psi(h..i))$ is stored in cache or in slow memory. Let

$$v \in B_r^{(ent)}(\psi(h..i)) = (B_r(\psi[1..h]) \cup O) \cap \psi(h..i),$$

where $O = I_R$ is the set of output vertices of G (also, input vertices of G_R). Let $v = \psi[g]$, with $h < g \leq i$.

Case 1. If $v \in O$, then at the time $\tau_g \in [\tau_i, \tau_h)$ when it has been visited v has been computed for the last time. Thereafter, by the rules of the I/O model, the value of v must be kept in memory till the end of the computation. At time τ_h , v can be either in cache or slow memory, but at some time $t > \tau_g$ it must be written in slow memory.

Case 2. If $v \in B_r(\psi[1..h])$, then $\psi[1..h]$ includes an r -enabler of v and we consider the smallest index f such that $\psi[1..f]$ includes an r -enabler of v . Clearly $f \leq h$ and $u = \psi[f]$ is a successor of v in G . We argue that the value of v must be in memory during the interval $(\tau_g, \tau_f]$. In fact, when u is computed, v must be in cache. We separately analyze two subcases.

Case 2.1. If $v \in V \setminus I$, then it is not computed at any time $t \in (\tau_g, \tau_f]$, otherwise v would be visited at t . Since $\tau_g < \tau_h \leq \tau_f$, we have that (a copy of the value of) v is in memory at time τ_h .

Case 2.2. If $v \in I$, then it is not read from slow memory at any time $t \in (\tau_g, \tau_f]$, otherwise v would be visited at t . Then, throughout this interval, v must be kept in cache. Since $\tau_g < \tau_h \leq \tau_f$, we have that (a copy of the value of) v is in cache at time τ_h .

Given that at most M vertices of $B_r^{(ent)}(\psi(h..i))$ can be in cache at τ_h , we conclude that at least $\max\{0, |B_r^{(ent)}(\psi(h..i))| - M\} = \max\{0, b_r^{ent}(\psi(h..i)) - M\}$ vertices must be in slow memory. Such vertices must all fall under Case 2.1 since the vertices in Case 2.2 must be in the cache. Then, they must have been written into slow memory, thus contributing to the number of write I/O, like the vertices in O (Case 1). Moreover, the contributions of the (disjoint) segments of partition \mathbf{i} can be added, since they count write operations involving vertices that belong to disjoint sets. This concludes the proof for (1).

Proof of (2) – Modified Boundary bound. By examining the argument for Case 2.1, we see that the vertices involved must also be read, at some time $t \geq \tau_h$, so that, those that are not in cache at time τ_h contribute to the number of read I/O. Each vertex in I is read at least once from slow memory. Finally, considering that no read I/O has been argued when $v \in O$, we reach (2).

Proof of (3) – Post-dominator bound. We claim that the set Y of vertices that are in cache at time τ_i or are read into the cache during the interval $[\tau_i, \tau_h)$ is a dominator set of $\psi(h..i)$ in G or, equivalently, a post-dominator set of $\psi(h..i)$ in G_R .

Let $v = \psi[g]$, with $h < g \leq i$. If $v \in I$, then at the time $\tau_g \in [\tau_i, \tau_h)$, when v has been visited, it has been read into the cache, so that $v \in Y$, whence v is dominated by Y .

If $v \in V \setminus I$, then at the time $\tau_g \in [\tau_i, \tau_h)$ when v has been visited it has been computed. If, by way of contradiction, v is not dominated by Y , then there is a directed path in G , say $(v_1, v_2, \dots, v_q = v)$, with no vertex in Y . Let v_s be the first vertex on this path computed during interval $[\tau_i, \tau_h)$, which must exist since v is computed during such interval. When v_s is computed, v_{s-1} must be in cache, since it is one of its operands. However, during $[\tau_i, \tau_h)$, v_{s-1} cannot be in cache since is not computed (by the definition of v_s), nor is it available at the initial time τ_i or read from slow memory (since $v_s \notin Y$). Thus, we have reached a contradiction, which shows that v is actually dominated by Y .

Given that at most M vertices of Y can be initially in the cache, we conclude that at least $\max\{0, |Y| - M\} \geq \max\{0, pd_{min}(\psi(h..i)) - M\}$ must be brought into cache by read operations that occur during the interval $[\tau_i, \tau_h)$. Moreover, the contributions of the (disjoint) segments of partition \mathbf{i} can be added since they count read operations occurring in different time intervals. This concludes the proof for (3).

Finally, a straightforward combination of bounds (1), (2), and (3) yields bound (4). ◀

We observe that, in Lemma 14, we can choose the visit rule and then, for the visit ψ corresponding to a given computation ϕ , we can choose the segment partition \mathbf{i} with the goal of maximizing the resulting lower bound for the cost metric of interest. However, for the lower bound to apply to (all computations of) DAG G , we have to consider the minimum lower bound over all visits. The preceding observations are made more formal in the next theorem. It is generally possible that none of the computations that minimize the number of read operations also minimizes the number of write operations, so that the I/O complexity may be larger than the sum of the read and of the write complexity.

► **Theorem 15** (I/O lower bound). *Given a DAG $G = (V, E)$, with input set I and output set O , a visit rule $r \in \mathcal{R}(G_R)$, a visit of G_R according to this rule $\psi \in \Psi_r(G_R)$, and a cache size M , we define the quantities*

$$\mathcal{W}_r(\psi, M) = \max_{\mathbf{i} \in \mathcal{I}(\psi)} \mathcal{W}_r(\mathbf{i}, \psi, M), \quad (5)$$

$$\mathcal{R}_r(\psi, M) = \max_{\mathbf{i} \in \mathcal{I}(\psi)} \mathcal{R}_r(\mathbf{i}, \psi, M). \quad (6)$$

where $\mathcal{I}(\psi)$ denotes the set of all segment partitions of ψ and the quantities $\mathcal{W}_\psi(\mathbf{i}, M)$ and $\mathcal{R}_\psi(\mathbf{i}, M)$ are those introduced in Equations (1) and (3), respectively. Then, the write I/O complexity $IO_{\mathcal{W}}(G, M)$, the read I/O complexity $IO_{\mathcal{R}}(G, M)$, and the total I/O complexity $IO(G, M)$ satisfy the following bounds:

$$IO_{\mathcal{W}}(G, M) \geq \min_{\psi \in \Psi_r(G_R)} \mathcal{W}_r(\psi, M), \quad (7)$$

$$IO_{\mathcal{R}}(G, M) \geq \min_{\psi \in \Psi_r(G_R)} \max\{\mathcal{R}_r(\psi, M), \mathcal{W}_r(\psi, M) + |I| - |O|\}, \quad (8)$$

$$IO(G, M) \geq \min_{\psi \in \Psi_r(G_R)} \{\mathcal{W}_r(\psi, M) + \max\{\mathcal{R}_r(\psi, M), \mathcal{W}_r(\psi, M) + |I| - |O|\}\}. \quad (9)$$

Proof. Bound (1) of Lemma 14 applies to a computation ϕ of G for which the procedure constructing the visit outputs ψ . The bound holds for any segment partition \mathbf{i} and, in particular, for the partition that maximizes $\mathcal{W}_r(\mathbf{i}, \psi, M)$. Using Definition (5), we obtain $IO_{\mathcal{W}}(\phi, M) \geq \mathcal{W}_r(\psi, M)$. To formulate a lower bound that holds for any computation ϕ , hence for any visit ψ , we need to minimize with respect to $\psi \in \Psi_r(G_R)$, arriving at Bound (7). Bounds (8) and (9) are established by analogous arguments. ◀

5.3 Comparison with Hong and Kung's S-partition technique

Hong and Kung [23] introduced the “*S-partition technique*”, for I/O lower bounds. In this section, we show that the central result of their approach can be derived as a corollary of the visit partition approach, when the latter is specialized to the topological visit rule $r^{(top)}$.

The *S*-partitions of a DAG are defined in terms of dominator and minimum sets. Given a DAG $G = (V, E)$ and a set $V' \subseteq V$, we say that $D \subseteq V$ is a *dominator set* of V' if every directed path from an input vertex of G to a vertex in V' intersects D . The *minimum set* of V' is the set of all vertices of V' that have no successors in V' . An *S*-partition is a sequence (V_1, V_2, \dots, V_k) of sets such that (a) they are disjoint and their union equals V ; (b) each V_j has a dominator set of size at most S ; (c) the minimum set of each V_j has size at most S ; (d) there is no edge from a vertex in V_j to a vertex in $\cup_{i=1}^{j-1} V_i$.

► **Theorem 16** (Adapted from [23, Theorem 3.1]). *Any computation of a DAG $G = (V, E)$ on the I/O model with a cache of M words, executing q I/O operations, is associated with a $2M$ -partition of G with k sets, such that $Mk > q > M(k-1)$. Therefore, if $k(G, 2M)$ is the minimum size of a $2M$ -partition of G , the I/O complexity of G satisfies:*

$$IO(G, M) \geq M(k(G, 2M) - 1). \quad (10)$$

Next, we show that when $r = r^{(top)}$, Bound (9) of Theorem 15 implies Bound (10) of Theorem 16. In the visit framework, minimum sets arise as the boundary of $r^{(top)}$ -visits.

► **Theorem 17** (Comparison with S-partition technique). *Given a DAG G , let ψ be any $r^{(top)}$ -visit of G_R . There exists at least one segment partition $\mathbf{i} = (i_1, \dots, i_k)$ of ψ such that*

$$\mathcal{W}_r(\mathbf{i}, \psi, M) + \mathcal{R}_r(\mathbf{i}, \psi, M) \geq M(k(G, 2M) - 1),$$

whence $IO(G, M) \geq M(k(G, 2M) - 1)$.

The proof of Theorem 17 is presented in the Appendix.

Diamond DAGs can be obtained by taking a $b \times b$ mesh (*i.e.*, a two-dimensional array) and by directing all the edges towards the upper right corner. The graph obtained as such has $n = b^2$ vertices, a single input vertex (*i.e.*, in the bottom left corner), and a single output vertex (*i.e.*, in the upper right corner). We show that for these DAGs, the visit partition technique yields tight I/O lower bounds, whereas the S -partition technique would only yield an $\Omega(1)$ lower bound.

Let $G = (V, E)$ be a b -side Diamond DAG. According to the definition of S -partition, the family $\{V\}$ is a 1-partition of V as V has a dominator (resp., minimum set) of size 1 composed by the single input (resp., output) vertex. Hence, for any $M \geq 1$, $k(G, 2M) = 1$ so that Hong and Kung's method in Theorem 16 ([23, Theorem 3.1]) yields a trivial bound. In contrast, the visit partitioning technique allows to obtain an asymptotically tight I/O lower bound:

► **Theorem 18** (I/O complexity Diamond DAG). *Let $G = (V, E)$ be a diamond DAG of side b . The I/O-complexity of G using a cache memory of size M satisfies $IO(G, M) \geq b/4M$.*

The proof of Theorem 18, an asymptotically matching upper bound, and an extension to a more general family of diamond DAGs are presented in the extended version [9, Section 5.4].

5.4 Extensions to related I/O models

External Memory Model. The result in Theorem 15 can be straightforwardly extended to the External Memory Model of Aggarwal and Vitter [1], where a single I/O operation can move $L \geq 1$ memory words between cache and consecutive slow-memory locations.

Models with asymmetric cost of read and write I/O operations. Since our method distinguishes the contribution of write and reads I/O operations, it would be interesting to use it to investigate I/O lower bounds where reads and writes have different cost [12, 20], including the case in which only the cost of write I/O operations is considered [2, 14]. To this end, the lower bound in (9) can be modified to include multiplicative scaling for each component.

Dropping the slow memory requirement for output values. By using a modified concept of r -entering boundary of a segment, defined as $\hat{B}_r^{(ent)}(\psi(i_{j-1}..i_j)) = B_r(\psi[1..i_{j-1}]) \cap \psi(i_{j-1}..i_j]$, our method yields I/O lower bounds in a modified version of the I/O models where output values are not required to be written into the slow memory.

Free-input model. The visit partition technique can also be adapted to the “*free input*” model, more akin to the pebbling model, in which input values can be generated into cache at any time (e.g., they are read from a dedicated ROM memory), rather than being initially stored in the slow memory. While our lower bound to the number of write I/O operations (7) remains unchanged, the lower bound to the number of read I/O operations (8) must be revised removing the contribution of the post-dominator bound and the read I/O term of the boundary-bound. For any $r \in \mathcal{R}(G_R)$ we have:

$$IO_{\mathcal{R}}^{fi}(G, M) \geq \min_{\psi \in \Psi_r(G_R)} \mathcal{W}_r(\psi, M) - |O|,$$

and, thus, $IO^{fi}(G, M) \geq \min_{\psi \in \Psi_r(G_R)} 2\mathcal{W}_r(\psi, M) - |O|$.

Execution with no recomputation. Finally, our result in Theorem 15 can be adapted and simplified to yield I/O lower bounds assuming that the value associated to any vertex $V \setminus I$ is computed exactly once (the no-recomputation assumption). This simplifying assumption is often of interest as it focuses the analysis on schedules with a minimum number of computational steps. Moreover, it may provide a stepping stone towards the analysis of the more general and challenging case where recomputation is allowed. Without recomputation, computational schedules correspond to the topological orderings of G . Thus, for any $r \in \mathcal{R}(G_R)$, the corresponding r -visits of G_R constructed according to the procedure discussed in Section 5.2, are the topological orderings of G_R . Therefore, we can restrict our attention to such orderings obtaining the following corollary:

► **Corollary 19.** *Given a DAG $G = (V, E)$, with input set I and output set O , consider its computations in the I/O model using a cache of size M such that no value is ever computed more than once. For any $r \in \mathcal{R}(G_R)$ we have:*

$$\begin{aligned} IO_{\mathcal{W}}^{nr}(G, M) &\geq \min_{\psi \in \Psi_{r(top)}(G_R)} \mathcal{W}_r(\psi, M), \\ IO_{\mathcal{R}}^{nr}(G, M) &\geq \min_{\psi \in \Psi_{r(top)}(G_R)} \max\{\mathcal{R}_r(\psi, M), \mathcal{W}_r(\psi, M) + |I| - |O|\}, \\ IO^{nr}(G, M) &\geq \min_{\psi \in \Psi_{r(top)}(G_R)} \{\mathcal{W}_r(\psi, M) + \max\{\mathcal{R}_r(\psi, M), \mathcal{W}_r(\psi, M) + |I| - |O|\}\}. \end{aligned}$$

The bounds obtained for computations without recomputation are generally higher than the general ones in Theorem 15, as while for each rule r we still analyze the entering boundary and the minimum post-dominator size of visit partitions, the set of visits to be considered is restricted to the subset $\Psi_{r(top)}(G_R) \subseteq \Psi_r(G_R)$, thus possibly eliminating some visit ψ with low $\mathcal{W}_r(\psi, M)$ and/or $\mathcal{R}_r(\psi, M)$. It is easy to see that the best lower bounds are obtained for the choice $r = r^{(sin)}$.

6 Conclusions

We have proposed the visit framework to investigate both space and I/O complexity lower bounds. The universal upper bounds we have obtained for both the singleton and the topological types of visits show that these types cannot yield tight pebbling lower bounds for all DAGs, although they do for some DAGs. The framework gives ample flexibility to tailor the type of visit to the given DAG, but we do not yet have good insights either on how to exploit this flexibility or on how to show that this flexibility is not ultimately helpful.

The spectrum of visit types exhibits the following tradeoff. As we go from the topological rule to, say, the singleton rule, by relaxing the enablement constraints, the set of vertex sequences that qualifies as a visit increases (which goes in the direction of reducing the boundary complexity of the DAG, since the minimization takes place over a larger domain), but the boundary complexity of a specific visit also increases (which goes in the direction of increasing the boundary complexity of the DAG, since the function to be minimized increases). The tension between these opposite forces has proven difficult to analyze quantitatively. The arguments used to establish universal upper bounds in the singleton and in the topological cases are significantly different, and it is not clear how to interpolate them for an intermediate visit type. Further research is clearly needed to make progress on what appears to be a rich combinatorial problem.

Another contribution of the visit framework is a step toward a unified treatment of pebbling and I/O complexity. Within the framework, we have already seen how to generalize the by now classical Hong-Kung partition technique, based on dominator and minimum sets, thus achieving much better lower bounds for some DAGs.

We conjecture that, for other significant DAGs whose I/O complexity cannot be well captured by the S -partition technique, visit partitions will help obtain good lower bounds. Good candidates are DAGs with a constant degree and a space complexity $S(N)$ superlinear, say polynomial, in the number of inputs $N = |I|$. For such DAGs, the size of the minimum dominator set cannot exceed N and, as shown in Theorem 17, the size of the topological boundary, *i.e.*, of the minimum set, is $O(\log N)$ at any point in the computation. On the other hand, the singleton boundary could be significantly higher.

Although we have not explicitly discussed the issue in this work, we do not have general I/O upper bounds matching our visit partition lower bounds. Therefore, further work is needed to achieve a full characterization of the I/O complexity of a DAG.

References

- 1 Alok Aggarwal and S. Vitter, Jeffrey. The Input/Output Complexity of Sorting and Related Problems. *Communications of the ACM*, 31(9):1116–1127, September 1988. doi:10.1145/48529.48535.
- 2 Grey Ballard, Dulceneia Becker, James Demmel, Jack Dongarra, Alex Druinsky, Inon Peled, Oded Schwartz, Sivan Toledo, and Ichitaro Yamazaki. Communication-avoiding symmetric-indefinite factorization. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1364–1406, 2014.
- 3 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Brief announcement: Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 77–79. ACM, 2012.
- 4 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing*, 32(6):3495–3523, 2010.
- 5 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.
- 6 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)*, 59(6):1–23, 2013.
- 7 Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Strassen’s matrix multiplication with recomputation. In *Workshop on Algorithms and Data Structures*, pages 181–192. Springer, 2017.
- 8 Gianfranco Bilardi and Lorenzo De Stefani. The I/O Complexity of Toom-Cook Integer Multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’19, pages 2034–2052, USA, 2019. Society for Industrial and Applied Mathematics.
- 9 Gianfranco Bilardi and Lorenzo De Stefani. The dag visit approach for pebbling and i/o lower bounds, 2022. doi:10.48550/arXiv.2210.01897.
- 10 Gianfranco Bilardi, Andrea Pietracaprina, and Paolo D’Alberto. On the space and access complexity of computation DAGs. In *Graph-Theoretic Concepts in Computer Science*, pages 47–58. Springer, 2000.
- 11 Gianfranco Bilardi and Franco Preparata. Processor-Time trade offs under bounded speed message propagation. Part 2: Lower Bounds. *Theory of Computing Systems*, 32(5):531–559, 1999.
- 12 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Efficient Algorithms with Asymmetric Read and Write Costs. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22–24, 2016, Aarhus, Denmark*, volume 57 of *LIPICs*, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.14.

- 13 Timothy Carpenter, Fabrice Rastello, P. Sadayappan, and Anastasios Sidiropoulos. Brief Announcement: Approximating the I/O Complexity of One-Shot Red-Blue Pebbling. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '16*, pages 161–163, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2935764.2935807.
- 14 Erin C. Carson, James Demmel, Laura Grigori, Nicholas Knight, Penporn Koanantakool, Oded Schwartz, and Harsha Vardhan Simhadri. Write-Avoiding Algorithms. In *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*, pages 648–658. IEEE Computer Society, 2016. doi:10.1109/IPDPS.2016.114.
- 15 Lorenzo De Stefani. *On space constrained computations*. PhD thesis, University of Padova, 2016.
- 16 Lorenzo De Stefani. Brief Announcement: On the I/O Complexity of Sequential and Parallel Hybrid Integer Multiplication Algorithms. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '22*, pages 449–452, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3490148.3538551.
- 17 Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. On characterizing the data access complexity of programs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 567–580, 2015.
- 18 Harvey Friedman. Algorithmic procedures, generalized turing algorithms, and elementary recursion theory. In *Studies in Logic and the Foundations of Mathematics*, volume 61, pages 361–389. Elsevier, 1971.
- 19 D. Yu Grigor'ev. Application of separability and independence notions for proving lower bounds of circuit complexity. *Zapiski Nauchnykh Seminarov POMI*, 60:38–48, 1976.
- 20 Yan Gu, Yihan Sun, and Guy E. Blelloch. Algorithmic building blocks for asymmetric memories. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 44:1–44:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.44.
- 21 John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM (JACM)*, 24(2):332–337, 1977.
- 22 Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.
- 23 Hong Jia-Wei and Hsiang-Tsung Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, 1981.
- 24 Lynn H Loomis and Hassler Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 55(10):961–962, 1949.
- 25 Auguste Olivry, Guillaume Iooss, Nicolas Tollenaere, Atanas Rountev, P. Sadayappan, and Fabrice Rastello. Ioopt: automatic derivation of I/O complexity bounds for affine programs. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 1187–1202. ACM, 2021. doi:10.1145/3453483.3454103.
- 26 Rasmus Pagh and Morten Stöckel. The input/output complexity of sparse matrix multiplication. In *European Symposium on Algorithms*, pages 750–761. Springer, 2014.
- 27 Michael S Paterson and Carl E Hewitt. Comparative schematology. In *Record of the Project MAC conference on concurrent systems and parallel computation*, pages 119–127. ACM, 1970.
- 28 Wolfgang J Paul, Robert Endre Tarjan, and James R Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10(1):239–251, 1976.

- 29 Desh Ranjan, John Savage, and Mohammad Zubair. Upper and lower I/O bounds for pebbling r -pyramids. *Journal of Discrete Algorithms*, 14:2–12, 2012.
- 30 J. E. Savage. Extending the Hong-Kung model to memory hierarchies. In *Computing and Combinatorics*, pages 270–281. Springer, 1995.
- 31 John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.
- 32 Jacob Scott, Olga Holtz, and Oded Schwartz. Matrix multiplication I/O-complexity by path routing. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 35–45, 2015.
- 33 Michele Scquizzato and Francesco Silvestri. Communication lower bounds for distributed-memory computations. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5–8, 2014, Lyon, France*, volume 25 of *LIPICs*, pages 627–638. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.STACS.2014.627.

A Proofs of technical results

Proof of Theorem 5. Consider a pebbling schedule ϕ of G which uses s pebbles, and any visit rule $r \in \mathcal{R}(G_R)$. The proof proceeds by constructing an r -visit ψ of G_R whose r -boundary complexity is itself bounded from above by s . Let T denote the total number of steps of the pebbling ϕ . The construction of ψ proceeds iteratively starting from the end of the pebbling schedule ϕ to its beginning: Let v denote the vertex which is being pebbled at the i -th step of ϕ for $1 \leq i \leq T$, then the same vertex v is visited in G_R if and only if all the vertices in at least one of the subsets in the enabling family $h(v)$ have already been visited. By construction, ψ is indeed a valid r -visit of G_R .

By the construction of the reverse DAG G_R , the set of the successors of any vertex $v \in V$ in G corresponds to the set of predecessors of v in G_R . By the rules of the pebble game, when considering a complete pebbling schedule for G , each vertex $v \in V$ is pebbled in for the first time before any of its successors. As each enabler in $r(v)$ is a subset of the predecessors of v in G_R and, hence, a subset of the successors of v in G , each vertex will surely be visited in ψ at the step corresponding to its first pebbling in ϕ unless it has already been visited. This allows us to conclude that all vertices of G_R are indeed visited by ψ .

Let $\phi[t]$ denote the vertex being pebbled at the t -th step of the pebbling schedule, for $1 \leq t \leq T$. In order to prove that the statement holds, it must be shown that, fixed an index i , $1 \leq i \leq n$ with $\psi[i] = \phi[t]$, for some $t \in \{1, 2, \dots, T\}$, the vertices in $B_r(\psi[1..i])$ must be pebbled (*i.e.*, held in the memory) at the end of t -th step of ϕ .

Let $v \in B_r(\psi[1..i])$. By the construction of ψ there must exist two indices t_1 and t_2 , with $1 \leq t_1 \leq t < t_2 \leq T$, such that $\phi[t_1] = v$, $\phi[t_2] \in Q \in r(v)$, and $\phi[t'] \neq v$ for every $t_1 < t' < t_2$ (if that was not the case, then v would have been visited then). As a consequence, the value of v computed at step t_1 of ϕ is used to compute v_{i_2} and therefore it must reside in memory at the end of step t (*i.e.*, it has to be pebbled). Since i was chosen arbitrarily, the same reasoning applies for all indices $i = 1, 2, \dots, n$. We can thus conclude that the maximum number of pebbles used by ϕ (and thus, the memory space used by ψ) is no less than $\max_{1 \leq i \leq n} |B_r(\psi[1..i])| = b_r(\psi)$.

The theorem follows by minimizing over all possible r -visits $\psi \in \Psi_r(G_R)$. ◀

Proof of Lemma 6. To prove the lemma, we construct a pebbling schedule ϕ for G , which corresponds to the r -visit ψ of G_R : the construction proceeds iteratively starting from the end of ψ . For any index $i = 1, 2, \dots, n$ let $\psi[i]$ denote the vertex visited at the i -th step of ψ .

Let G_n denote the sub-DAG of G induced by the subset of V composed of $\psi[n]$ and the set of its ancestors in G . The schedule ϕ starts following the steps of any pebbling schedule

of G_n . Once $\psi[n]$ has been pebbled, all the pebbles on vertices of G_n , except for $\psi[n]$, are removed. The schedule ϕ then proceeds according to the same procedure up to $\psi[1]$. During the i -th step, the pebbling schedule for G_i never re-pebbles any of the vertices in $\psi[i+1..n]$. As they were previously pebbled, we can assume that they maintain a pebble (*i.e.*, they are kept in memory) until the end of the computation. Hence it is possible to pebble G_i without re-pebbling the vertices in $\psi[i+1..n]$. By construction, ϕ is a complete pebbling of G .

Crucially, the order according to which the vertices are pebbled for the *last time* in ϕ corresponds to the reverse order of appearance of the vertices in the given visit ψ . When applying the conversion between pebbling schedules and visits discussed in the proof of Theorem 5, we have that the vertices are visited in G_R according to the order of their *last pebbling* in ϕ . The lemma follows. \blacktriangleleft

Proof of Theorem 9. We construct inductively an r -visit ψ of G such that $b_r(\psi) \leq \ell(d_{out} - 1)$.

In the base case $\ell = 0$, that is, all the vertices of G are input vertices without predecessors. Any permutation of the input vertices is an r -visit. As input vertices are enabled by the empty set, by Definition 3, they do not appear in the boundary and, thus, the r -boundary complexity of any such visit is zero.

For the general case $\ell \geq 1$, the visit begins by visiting any input vertex of G . If none of its direct successors are enabled according to r , by definition, $B_r(\psi[1]) = \emptyset$. If that is the case, the visit proceeds by selecting another input vertex of G .

Without loss of generality, let v denote the first input vertex of G whose r -enabled reach given the r -sequence ψ_{pre} constructed so far is not empty. As visiting v can only enable its at most d_{out} successors, we have $|B_r(\psi_{pre})| \leq d_{out}$. Let $G' = (V', E')$ denote the sub-DAG of G induced by the r -enabled reach of v given ψ_{pre} and let $r'(v) = \{Q \setminus \psi_{pre} \mid Q \in r(v)\}$ for all $v \in \text{reach}_r(v|\psi_{pre})$. By construction, G' is a sub-DAG of the DAG induced by the set of descendants of v , whose topological depth must be at most $\ell - 1$. Thus, G' has topological depth at most $\ell - 1$ as well. Hence, by the inductive hypothesis, for any visit rule of G' , and, in particular, for r' there exists an r' -visit of G' , denoted as ψ' such that $b_{r'}(\psi') \leq (\ell - 1)(d_{out} - 1) + 1$.

By Lemma 8, $\psi_{pre}\psi'$ is an r -sequence of G and vertices in ψ' do not enable any vertex in $V \setminus \psi'$. Further, as at the first step of ψ' one successor of v is visited, from that step onward, at most $d_{out} - 1$ successors of v which are yet to be visited may be in the boundary. Thus:

$$\begin{aligned} |B_r(\psi_{pre}[1..i])| &= 0 && \text{for } i = 1, \dots, |\psi_{pre}| \\ |B_r(\psi_{pre})| &\leq d_{out} \\ |B_r(\psi_{pre}\psi'[1..j])| &\leq (d_{out} - 1) + (\ell - 1)(d_{out} - 1) + 1 && \text{for } j = 1, \dots, |\psi'| \\ |B_r(\psi_{pre}\psi')| &\leq (d_{out} - 1) \end{aligned}$$

The visit then proceeds by visiting any input vertex of G which is yet to be visited and its enabled reach given the r -sequence constructed so far and by repeating the operations previously described. This ensures that G is entirely visited by ψ . By repeating the considerations on the boundary size previously discussed, we can conclude that the maximum boundary size of ψ is at most $(d_{out} - 1)\ell + 1$. The theorem follows. \blacktriangleleft

Proof of 12.

- (a) If $d_{out} = 0$, then all vertices of G are input vertices and any ordering ψ of V is a legal topological ordering, with $b_{r(top)}(\psi) = 0$, since input vertices are never part of the boundary.

- (b) If $d_{out} = 1$, then it is easy to see that G consists of a collection of chains (one for each input), at least one of which has length greater than 1. Clearly, visiting one chain at the time yields a visit ψ with $b_{r(top)}(\psi) = 1$.
- (c) If $d_{out} \leq D$, with $D \geq 2$, we proceed by induction on n . In the base case, $n = 1$, clearly there is a unique schedule ψ with $b_{r(top)}(\psi) = 0$. For the inductive step ($n > 1$), we recursively construct a visit $\psi = topv(G)$ as follows. Arbitrarily choose an input vertex u of G ; let $B_{r(top)}(u) = \{v_1, v_2, \dots, v_k\} \subseteq \text{suc}(u)$, with $0 \leq k \leq d_{out}(u) \leq D$; and output $\psi = uv_{j_1}\phi_1 \dots v_{j_k}\phi_k\phi$ where, informally, ϕ_h is a visit of the enabled reach of v_{j_h} , given the prefix of ψ to the left of v_{j_h} . Each v_{j_h} is chosen among the successors of u (initially) enabled by u and not yet visited so as to minimize the size of the enabled reach. Finally, ϕ is a visit of the vertices not yet traversed by the end of ϕ_k . More formally:
- j_1, \dots, j_k is a permutation of $1, \dots, k$, specified below.
 - For $h = 1, \dots, k$, $v_{j_h}\phi_h = topv(G_h)$, where G_h is the subgraph induced by $\{v_{j_h}\} \cup \text{reach}_{r(top)}(v_{j_h} | uv_{j_1}\phi_1 \dots v_{j_{h-1}}\phi_{h-1}v_{j_h})$. It is easy to see that any visit of G_h must begin with vertex v_{j_h} , the only input vertex of G_h , every other vertex of G_h being a proper descendant of v_{j_h} .
 - $j_h = \arg \min_{j \in \{1, \dots, k\} \setminus \{j_1, \dots, j_{h-1}\}} |\text{reach}_{r(top)}(v_j | uv_{j_1}\phi_1 \dots v_{j_{h-1}}\phi_{h-1}v_j)|$.
 - $\phi = topv(\bar{G})$, where \bar{G} is the subgraph induced by $V \setminus \{uv_{j_1}\phi_1 \dots v_{j_k}\phi_k\}$.

To establish the claimed bound on $b_{r(top)}(\psi)$, we make the following observations.

1. For $h = 1, \dots, k$, the boundary of the prefix of ψ ending just before v_{j_h} equals $\{v_{j_h}, \dots, v_{j_k}\}$, hence it has size $k - h + 1 \leq k$.
2. By Lemma 8, while visiting G_h , the boundary is at most $(k - h) + b_{r(top)}(v_{j_h}\phi_h)$.
3. Collectively, the $k - h + 1$ enabled reaches from which the next one to be visited is chosen contain at most $n - 1 - k$ vertices (because u and v_1, \dots, v_k are not contained in any of them). Since, by Lemma 11, these reaches are disjoint, the smallest ones contain at most $\lfloor \frac{n-1-k}{k-h+1} \rfloor$ vertices. Additionally accounting for v_{j_h} , G_h contains at most $1 + \lfloor \frac{n-1-k}{k-h+1} \rfloor = \lfloor \frac{n-h}{k-h+1} \rfloor$ vertices.
4. For convenience, let $f(n, D) = \frac{D-1}{\log_2 D} \log_2 n + 1$ and observe that f is increasing with both arguments. By the inductive hypothesis, we have:

$$b_{r(top)}(\psi) \leq \max\{k, \max_{h=1}^k \{(k-h) + f(\frac{n-h}{k-h+1}, D)\}, f(n-1-k, D)\}.$$

Using $f(\frac{n-h}{k-h+1}, D) \leq f(\frac{n}{k-h+1}, D)$ and letting $q = k - h + 1$, the previous yields

$$b_{r(top)}(\psi) \leq \max\{k, \max_{q=1}^k \{q-1 + f(\frac{n}{q}, D)\}, f(n-1-k, D)\}.$$

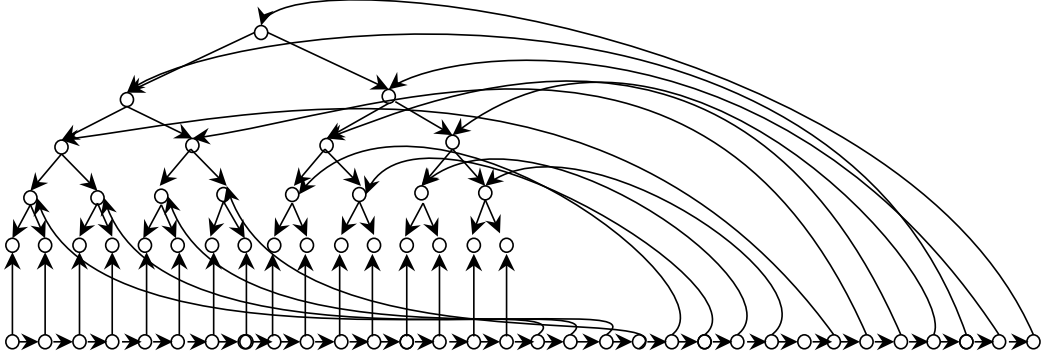
5. To complete the inductive step, it remains to show that each of the three terms in the outer max is no larger than $f(n, D)$. This is obvious for the third term $f(n-1-k, D)$, given the monotonicity of f . For the second term, we observe that each argument in the inner max satisfies

$$\begin{aligned} q-1 + f(\frac{n}{q}, D) &= q-1 + \frac{D-1}{\log_2 D} \log_2(\frac{n}{q}) + 1 \\ &= \left(\frac{D-1}{\log_2 D} \log_2 n + 1 \right) + (q-1) \left(1 - \frac{D-1}{\log_2 D} \frac{\log_2 q}{q-1} \right) \\ &\leq \frac{D-1}{\log_2 D} \log_2 n + 1, \end{aligned}$$

where the term dropped in the last step is negative or null, since $\frac{q-1}{\log_2 q} \leq \frac{D-1}{\log_2 D}$, for $q = 1, \dots, D$, as can be shown by straightforward calculus. Finally, to bound the first term, k , we observe that, if $k \leq 1$, then the target bound trivially holds. Otherwise,

consider that $n \geq k + 1$ (G contains at least the distinct vertices u and v_1, \dots, v_k) and that, for $2 \leq k \leq D$, we have $\frac{k-1}{\log_2 k} \leq \frac{D-1}{\log_2 D}$ (as mentioned above). Then, the target inequality follows:

$$k = (k - 1) + 1 \leq \frac{k - 1}{\log_2 k} \log_2 k + 1 < \frac{D - 1}{\log_2 D} \log_2 n + 1. \quad \blacktriangleleft$$



■ **Figure 1** This DAG allows us to study the relationship between the $r^{(sin)}$ -boundary complexity and the $r^{(top)}$ -boundary complexity. The DAG is obtained by connecting the vertices of a directed binary arborescence with a directed chain. Its $r^{(sin)}$ -boundary complexity is 2: consider a visit starting in the leftmost vertex of the chain and that visits vertices of the tree as soon as they are enabled. Instead, its $r^{(top)}$ -boundary complexity is $\log_2(n + 2) - 1$, where n denotes the number of its vertices. In every possible $r^{(top)}$ -visit, the directed chain must be visited first, and then the arborescence is left to be visited. The bound on the $r^{(top)}$ -boundary complexity follows an argument similar to that in the analysis of complete q -trees in the extended version of this work [9]. Consider the reverse DAG. Its pebbling number is $(n + 2) - 1$, where n denotes the number of vertices. (The proof of this statement is a simple exercise.) By the previous considerations, while using the $r^{(sin)}$ yields in Theorem 5 yields a trivial lower bound to the pebbling number. Instead, using $r^{(top)}$ yields a tight lower bound to the pebbling number of the DAG.

Proof of Theorem 17. For notational simplicity, throughout this proof, r stands for $r^{(top)}$. We preliminarily observe that if $\psi \in \Psi_{r^{(top)}}(G_R)$, then, for any segment $\psi(h, i]$, $B_r^{(ent)}(\psi(h, i])$ is the minimum set of $\psi(h, i]$ in G . In fact, by the definition of $r^{(top)}$, $v \in B_r(\psi[1, h])$ if and only if all of its predecessors in G_R (i.e., its successors in G) are in $\psi[1..h]$. Hence, $B_r^{(ent)}(\psi(h..i])$ contains exactly those vertices in $\psi(h, i]$ which are either inputs of G_R (outputs of G) or for which $\psi(h, i]$ contains no predecessor in G_R (thus, no successor in G).

Below we will make use of the following two properties, whose simple proof is omitted here. Let $U \subseteq V$ and $x \in V$. Then $min_{pd}(U \cup \{x\}) \leq min_{pd}(U) + 1$ and $sms(U \cup \{x\}) \leq sms(U) + 1$, where $sms(X)$ denotes the size of the minimum set of $X \subseteq V$.

Given $\psi \in \Psi_{r^{(top)}}(G_R)$, we consider a segment partition $\mathbf{i} = (i_1, i_2, \dots, i_k)$ where each segment, with the possible exception of the last one, has a minimum postdominator or the minimum set of size $2M$. Based on the properties stated in the preceding paragraph, such a partition can be easily constructed by scanning the visit one vertex at a time and closing a segment as soon as the desired condition is met, or all vertices have been scanned.

For $j = 1, \dots, k$, let $V_j = \{\psi[i_{k-j} + 1], \dots, \psi[i_{k+1-j}]\}$. We show that sequence (V_1, \dots, V_k) is a $2M$ -partition of G , by proving the defining properties: (a) As they correspond to the segments of a visit, the V_j 's are disjoint and their union equals V . (b) By construction, the

dominator of each V_j in G has size at most $2M$. (c) By construction, the minimum set of each V_j in G has size at most $2M$. (d) Finally, since ψ is an $r^{(top)}$ -visit of G_R , it is a reverse topological ordering of the vertices in G . Therefore, there are no edges of G from V_j to $\cup_{i=1}^{j-1} V_i$. Clearly, $k \geq k(G, 2M)$, by the definition of the latter quantity.

To analyze the I/O requirements of ψ , consider the following sequence of inequalities:

$$\begin{aligned}
\mathcal{W}_r(\psi, M) + \mathcal{R}_r(\psi, M) &\geq \mathcal{W}_r(\mathbf{i}, \psi, M) + \mathcal{R}_r(\mathbf{i}, \psi, M) \\
&\geq \sum_{j=1}^k \max\{0, b_r^{(ent)}(\psi(i_{j-1}..i_{j+1})) - M, \min_{pd}(\psi(i_{j-1}..i_j)) - M\} \\
&\geq \sum_{j=1}^{k-1} (2M - M) = (k-1)M \\
&\geq M(k(G, 2M) - 1).
\end{aligned}$$

These four inequalities respectively take into account (i) the definitions in (5) and (6); (ii) the definitions in (1) and (3); (iii) the fact that, for $j = 1, \dots, k-1$, at least one of the arguments of the max operator equals M ; and (iv) the relationship $k \geq k(G, 2M)$, seen above. Finally, since the chain of inequalities applies to any visit $\psi \in \Psi_{r^{(top)}}(G_R)$, we can invoke inequality (9) to conclude that $IO(G, M) \geq M(k(G, 2M) - 1)$. ◀

Counting and Sampling from Substructures Using Linear Algebraic Queries

Arijit Bishnu ✉

Indian Statistical Institute, Kolkata, India

Arijit Ghosh ✉

Indian Statistical Institute, Kolkata, India

Gopinath Mishra¹ ✉

University of Warwick, Coventry, UK

Manaswi Paraashar¹ ✉

Aarhus University, Denmark

Abstract

For an unknown $n \times n$ matrix A having non-negative entries, the *inner product* (IP) oracle takes as inputs a specified row (or a column) of A and a vector $\mathbf{v} \in \mathbb{R}^n$ with non-negative entries, and returns their inner product. Given two input vectors \mathbf{x} and \mathbf{y} in \mathbb{R}^n with non-negative entries, and an unknown matrix A with non-negative entries with IP oracle access, we design almost optimal sublinear time algorithms for the following two fundamental matrix problems:

- Find an estimate \mathcal{X} for the *bilinear form* $\mathbf{x}^T A \mathbf{y}$ such that $\mathcal{X} \approx \mathbf{x}^T A \mathbf{y}$.
- Designing a *sampler* \mathcal{Z} for the entries of the matrix A such that $\mathbb{P}(\mathcal{Z} = (i, j)) \approx x_i A_{ij} y_j / (\mathbf{x}^T A \mathbf{y})$, where x_i and y_j are i -th and j -th coordinate of \mathbf{x} and \mathbf{y} respectively.

As special cases of the above results, for any submatrix of an unknown matrix with non-negative entries and IP oracle access, we can efficiently estimate the sum of the entries of any submatrix, and also sample a random entry from the submatrix with probability proportional to its weight. We will show that the above results imply that if we are given IP oracle access to the adjacency matrix of a graph, with non-negative weights on the edges, then we can design sublinear time algorithms for the following two fundamental graph problems:

- Estimating the sum of the weights of the edges of an induced subgraph, and
- Sampling edges proportional to their weights from an induced subgraph.

We show that compared to the classical LOCAL queries (degree, adjacency, and neighbor queries) on graphs, we can get a quadratic speedup if we use IP oracle access for the above two problems.

Apart from the above, we study several matrix problems through the lens of IP oracle, like testing if the matrix is diagonal, symmetric, doubly stochastic, etc. Note that IP oracle is in the class of linear algebraic queries used lately in a series of works by Ben-Eliezer et al. [SODA'08], Nisan [SODA'21], Rashtchian et al. [RANDOM'20], Sun et al. [ICALP'19], and Shi and Woodruff [AAAI'19]. Recently, IP oracle was used by Bishnu et al. [RANDOM'21] to estimate dissimilarities between two matrices.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms

Keywords and phrases Query complexity, Bilinear form, Uniform sampling, Weighted graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.8

Related Version The missing proofs can be found in the full version of the paper.

Full Version: <https://arxiv.org/abs/2201.04975> [13]

¹ The work was done when the author was at Indian Statistical Institute, Kolkata, India.



1 Introduction

Let A be an unknown $n \times n$ matrix with oracle access and with non-negative entries. Consider a submatrix B of A given by two subsets S_1 and S_2 of indices of the rows and columns A , that is, $B := (A_{ij})_{(i,j) \in S_1 \times S_2}$. Note that submatrix B is unknown as the matrix A itself is unknown. We are only given as inputs the sets S_1 and S_2 , and oracle access to the unknown matrix A , and we have to *efficiently* solve the following two problems:

- Estimate the sum of the entries in the submatrix B , and
- sample a random entry (i, j) from B proportional to its weight A_{ij} , that is, we output $(i, j) \in S_1 \times S_2$ with probability

$$\frac{A_{ij}}{\sum_{(i,j) \in S_1 \times S_2} A_{ij}}.$$

We will show that using IP oracle access to A , which takes as inputs a specified row (or a column) of A and a vector $\mathbf{v} \in \mathbb{R}^n$ with non-negative entries and returns their inner product (see Section 1.1 for a formal definition), we can design efficient algorithms for the above defined fundamental problems. In fact, we will be giving efficient algorithms for even more general problems, see Section 1.2. Now, we discuss two consequences of our result on the above-mentioned matrix problem.

Counting and sampling edges from induced subgraphs

Let $G = (V(G), E(G))$ be an *unknown graph*¹ on n vertices, and S be any subset of vertices of G . Assume that the query access to graph G is INDUCED DEGREE query oracle which takes a vertex $u \in V(G)$ and a subset $X \subseteq V$ as input and reports the number of neighbors of u that are present in X . Given INDUCED DEGREE query access to G , two natural questions to consider on the subgraph G_S of G induced by the subset S are the following

- estimate the number of edges in G_S , and
- uniformly sample a random edge from G_S .

Observe that both of these graph problems with INDUCED DEGREE query access can be reduced to the matrix problems with IP oracle access to the adjacency matrix of the graph, where we only ask for inner product with the vectors in $\{0, 1\}^n$.

Weighted edge counting and sampling from graphs

For a graph with non-negative weights on its edges, we can similarly consider the problems of estimating the sum of the weights of the edges in the graph, and sampling edges of the graph proportional to its weight. Again, observe that these two fundamental problems on weighted graphs can also be reduced to the matrix problem defined at the beginning of this section by considering their adjacency matrix.

Notation

In this paper, we denote the set $\{1, \dots, t\}$ by $[t]$ and $\{0, \dots, t\}$ by $[[t]]$. For a (directed) graph G , $V(G)$ and $E(G)$ denote the vertex set and edge sets of G , we will use V and E when the graph is clear from the context. For a vertex u , let $d_G(u)$ denote the degree of u in G and $N_G(u)$ denote the set of neighbors of u in G . For a subset S of $V(G)$, the subgraph of G induced by S is denoted by $G_S = (S, E_S)$ such that $E_S := \{\{u, v\} \in E(G) \mid u \in S \text{ and } v \in S\}$. The LOCAL queries for a graph $G = (V(G), E(G))$ are:

¹ By unknown we mean that the vertex set $V(G)$ of G is known but the edge set $E(G)$ is not known.

DEGREE query: given $u \in V(G)$, the oracle reports the degree of u in $V(G)$

NEIGHBOR query: given $u \in V(G)$ and an integer i , the oracle reports the i -th neighbor of u , if it exists; otherwise, the oracle reports \perp ².

ADJACENCY query: given $u, v \in V(G)$, the oracle reports whether $\{u, v\} \in E(G)$.

For a non-empty set X and a given parameter $\varepsilon \in (0, 1)$, an *almost uniform* sample of X means each element of X is sampled with probability values that lie in the interval $[(1 - \varepsilon)/|X|, (1 + \varepsilon)/|X|]$. For a matrix A , A_{ij} denotes the element in the i -th row and j -th column of A . A_{i*} and A_{*j} denote the i -th row vector and j -th column vector of the matrix A , respectively. $A \in [[\rho]]^{n \times n}$ means $A_{ij} \in [[\rho]]$ for each $i, j \in [n]$, $\rho \in \mathbb{N}$. Throughout this paper, the number of rows or columns of a square matrix A is n , which will be clear from the context. Vectors are matrices of order $n \times 1$ and will be represented using boldface letters. Without loss of generality, we consider n to be a power of 2. The i -th coordinate of a vector \mathbf{x} is denoted by x_i . We denote by $\mathbf{1}$ the vector with all coordinates 1. Let $\{0, 1\}^n$ be the set of n -dimensional vectors with entries either 0 or 1. For $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{1}_{\mathbf{x}}$ is a vector in $\{0, 1\}^n$ whose i -th coordinate is 1 if $x_i \neq 0$ and 0 otherwise; $\text{nnz}(\mathbf{x}) = |\{i \in [n] : x_i \neq 0\}|$ denotes the number of non-zero components of the vector. By $\langle \mathbf{x}, \mathbf{y} \rangle$, we denote the standard inner product of \mathbf{x} and \mathbf{y} , that is, $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i$. For $a, b \in \mathbb{R}$, we say $a = (1 \pm \varepsilon)b$ if $a \in [(1 - \varepsilon)b, (1 + \varepsilon)b]$. Similarly, P is a $(1 \pm \varepsilon)$ -approximation to Q means $|P - Q| \leq \varepsilon \cdot Q$. *With high probability* means that the probability of success is at least $1 - \frac{1}{n^c}$, where c is a positive constant. $\tilde{\Theta}(\cdot)$ and $\tilde{O}(\cdot)$ hides a poly $(\log n, 1/\varepsilon)$ term in the upper bound.

1.1 Definition and motivation of INNER PRODUCT oracle

Let $A \in [[\rho]]^{n \times n}$, $\rho \in \mathbb{N}$, be a matrix whose size is known but the entries are unknown. Now given a row index $i \in [n]$ (or, a column index $j \in [n]$) and a vector $\mathbf{v} \in \mathbb{R}^n$, with non-negative entries as input, the INNER PRODUCT query to A reports the value of $\langle A_{i*}, \mathbf{v} \rangle$ ($\langle A_{*j}, \mathbf{v} \rangle$). If the input index is for row (column), we refer to the corresponding query as row (column) IP query. Observe that INDUCED DEGREE query can be implemented for a graph G by IP oracle as a dot product with $\mathbf{1}_S$ (indicator vector for the set S) and the corresponding row of the matrix A that is the 0/1 adjacency matrix of G .

The IP query has both graph theoretic and linear algebraic flavors to it and we will highlight them shortly. It may be mentioned here that IP has been already used to estimate the Hamming distance between two matrices [12].

From a practical point of view, Rashtchian et al. [32] mention that vector-matrix-vector queries would most likely be useful in the context of specialized hardware or distributed environments. Needless to say, the same carries over to IP query. There are many computer architectures that allow us to compute inner products in one cycle of computation with more parallel processors. Inner product computation can be parallelized using *single instruction multiple data* (SIMD) architecture [26]. Modern day GPU processors use instruction-level parallelism. Nvidia GPUs precisely do that by providing a single API call to compute inner products [35, 2]. There are many such architectures where IP query has been given to users directly. Similarly, there are programming language constructs built on SIMD framework that can compute inner products [1].

² The ordering of neighbors of the vertices are unknown to the algorithm.

1.2 The problems, results and paper organization

The main matrix-related problems considered in this work involve estimating the bilinear form $\mathbf{x}^T \mathbf{A} \mathbf{y}$ and sampling an element of a matrix almost uniformly using IP queries. Bilinear form estimation is inherently interesting as it is a generalization of the problem defined at the beginning of Section 1. Also, bilinear form estimation has huge importance in numerical linear algebra because of its use in calculating node centrality measures like resolvent subgraph centrality and resolvent subgraph communicability [10, 22], Katz score for adapting it to PageRank computing [14], etc.

BILINEAR FORM ESTIMATION_A(\mathbf{x}, \mathbf{y}), in short BFE_A(\mathbf{x}, \mathbf{y})

Input: Vectors $\mathbf{x} \in [[\gamma_1]]^n$, $\mathbf{y} \in [[\gamma_2]]^n$, IP access to matrix $A \in [[\rho]]^{n \times n}$, and $\varepsilon \in (0, 1)$.

Output: An $(1 \pm \varepsilon)$ -approximation to $\mathbf{x}^T \mathbf{A} \mathbf{y}$.

SAMPLE ALMOST UNIFORMLY_A(\mathbf{x}, \mathbf{y}), in short SAU_A(\mathbf{x}, \mathbf{y})

Input: Vectors $\mathbf{x} \in [[\gamma_1]]^n$, $\mathbf{y} \in [[\gamma_2]]^n$, IP access to matrix $A \in [[\rho]]^{n \times n}$, and $\varepsilon \in (0, 1)$.

Output: Report Z satisfying $(1 - \varepsilon) \frac{x_i A_{ij} y_j}{\mathbf{x}^T \mathbf{A} \mathbf{y}} \leq \mathbb{P}(Z = (i, j)) \leq (1 + \varepsilon) \frac{x_i A_{ij} y_j}{\mathbf{x}^T \mathbf{A} \mathbf{y}}$.

For the above problems, our results are stated in Theorem 1.1. We give the sketches of the proofs of the upper bound for a special case in Section 4. The general upper bounds can be derived from the special case and their proof is in the full version [13] of the paper. The lower bound parts of Theorem 1.1 are also proved in the full version [13].

► **Theorem 1.1.** BFE_A(\mathbf{x}, \mathbf{y}) and SAU_A(\mathbf{x}, \mathbf{y}) can be solved by using

$$\tilde{\Theta} \left(\frac{\sqrt{\rho \gamma_1 \gamma_2} (\text{nnz}(\mathbf{x}) + \text{nnz}(\mathbf{y}))}{\sqrt{\mathbf{x}^T \mathbf{A} \mathbf{y}}} \right)$$

IP queries, where $\text{nnz}(\mathbf{x})$ and $\text{nnz}(\mathbf{y})$ denotes the number of non-zero entries in \mathbf{x} and \mathbf{y} , respectively.

The above theorem has several important consequences. An immediate consequence of Theorem 1.1 is the following.

► **Corollary 1.2** (Estimating and sampling from submatrices of 0/1 matrix).³ Let A with $n \times n$ unknown 0/1 matrix with IP oracle access. Let $\varepsilon \in (0, 1)$, and S_1 and S_2 be subsets of indices of the rows and columns of the matrix A , respectively. For the matrix $B = (A_{ij})_{i \in S_1, j \in S_2}$ using $\tilde{\Theta} \left(\frac{|S_1| + |S_2|}{\sqrt{\text{nnz}(B)}} \right)$ IP queries, we can find an estimate \mathcal{X} of the number of ones in B such that $\mathcal{X} = (1 \pm \varepsilon) \sum_{(i,j) \in S_1 \times S_2} A_{ij}$, and also design a sampler \mathcal{Z} of $S_1 \times S_2$ satisfying, for all $(i, j) \in S_1 \times S_2$,

$$\mathbb{P}(\mathcal{Z} = (i, j)) = (1 \pm \varepsilon) \frac{A_{ij}}{\text{nnz}(B)}.$$

Furthermore, the IP queries used by both algorithms only ask for inner products with vectors in $\{0, 1\}^n$.

³ Even though Corollary 1.3 is more general than Corollary 1.2, we state Corollary 1.2 to show its application in Remark 1.4 (i).

We now define a more general class of matrices. Let $\mathcal{F}(n, \rho)$ be a family of $n \times n$ matrices with non-negative entries such that for all $A \in \mathcal{F}(n, \rho)$ we have

$$\min_{(i,j):A_{ij}>0} A_{ij} \geq 1 \quad \text{and} \quad \max_{(i,j):A_{ij}>0} A_{ij} \leq \rho.$$

The following result, which is a direct corollary from Theorem 1.1, shows that we can efficiently estimate and sample even from this family.

► **Corollary 1.3** (Estimating and sampling from an arbitrary matrix). *Let A with unknown matrix in $\mathcal{F}(n, \rho)$ with IP oracle access. Let $\varepsilon \in (0, 1)$, and S_1 and S_2 be subsets of indices of the rows and columns of the matrix A , respectively. For the matrix $B = (A_{ij})_{i \in S_1, j \in S_2}$ using*

$\tilde{O}\left(\frac{\sqrt{\rho}(|S_1|+|S_2|)}{\sqrt{\sum_{i,j} A_{ij}}}\right)$ IP queries, we can find an estimate \mathcal{X} of the number of ones in B , that is, $\mathcal{X} = (1 \pm \varepsilon) \sum_{i,j} B_{ij}$, and we can also design a sampler \mathcal{Z} of $[n] \times [n]$ satisfying, for all $(i, j) \in S_1 \times S_2$,

$$\mathbb{P}(\mathcal{Z} = (i, j)) = (1 \pm \varepsilon) \frac{A_{ij}}{\sum_{(i,j) \in S_1 \times S_2} A_{ij}}.$$

We would like to point out that the above corollaries are also tight because the lower bound part of the proof of Theorem 1.1 holds even for their special cases.

► **Remark 1.4** (Consequences of Theorem 1.1).

(i) **Induced subgraphs:** Given INDUCED DEGREE query access to a unknown graph $G = (V(G), E(G))$ and a subset $S \subset V(G)$ and an $\varepsilon \in (0, 1)$ as input, one can estimate the number edges in G_S up to $(1 \pm \varepsilon)$ factor by using $\tilde{O}\left(\frac{|S|}{\sqrt{m_S}}\right)$ queries and sample a random edge almost uniformly from G_S , where G_S denotes the subgraph of G induced by S . We can also design a sampler for the edges in G_S where each edge $e \in E(G_S)$ gets sampled with probability $(1 \pm \varepsilon) \frac{1}{|E(G_S)|}$ using the same number of queries. Note that this result follows from Corollary 1.2 as each INDUCED DEGREE query to graph G is analogous to an IP query to the adjacency matrix of G . It is because the algorithms corresponding to Corollary 1.2 only ask for inner product with vectors in $\{0, 1\}^n$.

(ii) **Weighted graphs:** Given IP oracle access to the adjacency matrix $A \in [[\rho]]^{n \times n}$ ⁴ of weighted graph $G = (V(G), E(G))$, then we can estimate $\omega(G) := \sum_{(i,j) \in E(G)} A_{ij}$ up to $(1 \pm \varepsilon)$ -approximation using at most $\tilde{O}\left(\frac{\sqrt{\rho n}}{\sqrt{\omega(G)}}\right)$ IP queries. We can also design a sampler for the edges of the graph G where each edge $(i, j) \in E(G)$ gets sampled with probability $(1 \pm \varepsilon) \frac{A_{ij}}{\omega(G)}$ using the same number of queries. Note that this result follows from Corollary 1.3, and is a generalization of the edge estimation results using local queries by Feige [23], and Goldreich and Ron [25].

(iii) **Stochastic matrices:** Let A be an unknown *doubly stochastic*⁵ $n \times n$ matrix with IP oracle access, and each non-zero entry of A is at least $\lambda > 0$. We can design a sampler \mathcal{Z} of $[n] \times [n]$ such that, for all $(i, j) \in [n] \times [n]$, we have $\mathbb{P}(\mathcal{Z} = (i, j)) = (1 \pm \varepsilon) A_{ij}$. Our sampler will use at most $\tilde{O}\left(n/\sqrt{\lambda}\right)$ queries.

⁴ Assume that G is a complete graph such that the weights on $\{i, j\} \notin E(G)$ is 0.

⁵ A matrix A with non-negative entries is *doubly stochastic* if the sum of the entries of any row or column is one.

In Section 5, we discuss several other matrix problems using IP oracle query which were studied using stronger queries like *matrix vector* and *vector matrix vector* queries [39, 32].

In Section 3 we establish that LOCAL query access (to the entire unknown graph) cannot solve problems in induced subgraphs *efficiently*. Two crucial takeaways are that IP oracle and its derivative, the INDUCED DEGREE oracle act like a local query on any induced subgraphs, and there is a quadratic separation between the powers of LOCAL query and INDUCED DEGREE queries to solve EDGE ESTIMATION and EDGE SAMPLING in induced subgraphs.

2 Inner product oracle vis-a-vis other query oracles

Graph parameter estimation, where the graph can be accessed through query oracles only, has been an active area of research in sub-linear algorithms for a while [25, 19, 20, 34]. There are different granularities at which the graph can be accessed – the query oracle can answer properties about graphs that are local or global in nature. By now, the LOCAL queries have been used for edge [25], triangle [19], clique estimation [20] and has got a wide acceptance among researchers. Apart from the local queries, in the last few years, researchers have also used the RANDOM EDGE query [4, 6], where the oracle returns an edge in the graph G uniformly at random. Notice that the randomness will be over the probability space of all edges, and hence, it is difficult to classify a random edge query as a local query. On the other hand, *global queries* come in different forms. Starting with the subset queries [37, 38, 33], there have been other queries like BIPARTITE INDEPENDENT SET query, INDEPENDENT SET query [8], GPIS query [11, 17], CUT query [34], etc. Linear measurements or queries [5, 3], based on dot product, have been used for different graph problems.

To this collection of query oracles, we introduce a new oracle called INNER PRODUCT (IP) oracle which is a natural oracle to consider for linear algebraic and graph problems. Using this oracle, we solve hitherto unsolved problems (by an unsolved problem, we mean that no non-trivial algorithm was known before) with graph theoretic and linear algebraic flavor, like (a) edge estimation in induced sub-graph; (b) bilinear form estimation; (c) sampling entries of matrices with non-negative entries. We also show weighted edge estimation and edge estimation in induced subgraph as applications of bilinear form estimation. Our lower bound result, for EDGE ESTIMATION in induced subgraph with only LOCAL query access, implies that there is a separation between the powers of LOCAL query and INDUCED DEGREE query. We will show that our newly introduced INNER PRODUCT query oracle can solve problems that can not be solved by the three local queries mentioned even coupled with the random edge query.

Our current survey of the literature (here we do not claim exhaustivity!) shows that a query related to a subgraph was first used in Ben-Eliezer et al. [9], and named as *group query*, where one asks if there is at least one edge between a vertex and a set of vertices. We found the latest query in this league to be the *demand query* (in bipartite graphs where the vertex set are partitioned into two parts left vertices and right vertices) introduced by Nissan [29] – a demand query accepts a left vertex and an order on the right vertices and returns the first vertex in that order that is a neighbor of the left vertex. One can observe that the group and demand queries are polylogarithmically equivalent. Staying on this line of study related to the relation of a vertex with a subset of vertices, we focus on the INDUCED DEGREE query which we feel handles many natural questions.

Query oracle based graph algorithms access the graph at different granularities – this gives rise to a whole gamut of queries with different capacities, ranging from local queries like degree, neighbor, adjacency queries [23, 25] to global queries like independent set based

queries [8, 17], random edge queries [4], and others like group [9] and demand queries [29]. This rich landscape of queries has unravelled many interesting algorithmic and complexity theoretic results [23, 25, 4, 9, 29, 8, 17, 33]. With this in mind, if we turn our focus to the landscape of linear algebraic queries, the most natural query is the *matrix entry* query where one gives an index of the matrix and asks for the value there. Lately, a series of works [32, 39, 36, 7] have used linear algebraic queries like *vector-matrix-vector query* and *matrix-vector* queries. The IP oracle is also motivated by these new query oracles. Notice the huge difference in power between *matrix entry* query and *vector-matrix-vector query* and *matrix-vector* queries. Note that IP query is strictly weaker than these matrix queries but stronger than the *matrix entry* query. We feel there is a need to study linear algebraic queries with intermediate power – the IP query fits in that slot.

3 A query model for induced subgraph problems

To the best of our knowledge, our work is a first attempt towards solving estimation problems in induced subgraphs. We start by showing a separation between LOCAL query and INDUCED DEGREE query using the problems of EDGE ESTIMATION and EDGE SAMPLING in induced subgraph. We now define INDUCED EDGE ESTIMATION and INDUCED EDGE SAMPLING.

INDUCED EDGE ESTIMATION

Input: A parameter $\varepsilon \in (0, 1)$ and a subset S of the vertex set V of a graph G .

Output: A $(1 \pm \varepsilon)$ -approximation to the number of edges E_S in the induced subgraph.

INDUCED EDGE SAMPLING

Input: A parameter $\varepsilon \in (0, 1)$ and a subset S of the vertex set V of a graph G .

Output: Sample each edge $e \in E_S$ with probability between $\frac{1-\varepsilon}{|E_S|}$ and $\frac{1+\varepsilon}{|E_S|}$.

One of the main contributions of this paper is to show that LOCAL queries together with RANDOM EDGE query are *inefficient* for both INDUCED EDGE ESTIMATION and INDUCED EDGE SAMPLING. The lower bound results follow.

► **Theorem 3.1** (Lower bound for INDUCED EDGE ESTIMATION using LOCAL queries). *Let us assume that $s, m_s \in \mathbb{N}$ be such that $1 \leq m_s \leq \binom{s}{2}$ and the query algorithms have access to DEGREE, NEIGHBOR, ADJACENCY and RANDOM EDGE queries to an unknown graph $G = (V(G), E(G))$. Any query algorithm that can decide for all $S \subseteq V(G)$, with $|S| = \Theta(s)$, whether $|E_S| = m_s$ or $|E_S| = 2m_s$, with probability at least $2/3$, requires $\Omega\left(\frac{s^2}{m_s}\right)$ queries.*

► **Theorem 3.2** (Lower bound for INDUCED EDGE SAMPLING using LOCAL QUERIES). *Let us assume that $s, m_s \in \mathbb{N}$ be such that $1 \leq m_s \leq \binom{s}{2}$ and the query algorithms have access to DEGREE, NEIGHBOR, ADJACENCY and RANDOM EDGE queries to an unknown graph $G = (V(G), E(G))$. Any query algorithm that for any $S \subseteq V(G)$, with $|S| = \Theta(s)$, samples the edges in E_S ε -almost uniformly⁶, with probability at least $99/100$, will require $\Omega\left(\frac{s^2}{m_s}\right)$ queries⁷. Note that $\varepsilon \in (0, 1)$ is given as an input to the algorithm.*

⁶ Each edge in E_S is sampled with probability between $(1 - \varepsilon)\frac{1}{|E_S|}$ and $(1 + \varepsilon)\frac{1}{|E_S|}$.

⁷ Let U denote the uniform distribution on E_S . The lower bound even holds even if the goal is to get a distribution that is ε close to U with respect to ℓ_1 distance.

► **Remark 3.3.** When $S = V$, INDUCED EDGE ESTIMATION and INDUCED EDGE SAMPLING are EDGE ESTIMATION and EDGE SAMPLING problems, respectively. Both EDGE ESTIMATION and EDGE SAMPLING can be solved with high probability by using $\tilde{\Theta}\left(|V|^2/|E|\right)$ ADJACENCY queries [24]. Notice that these bounds match the lower bounds. Contrast this with the fact that EDGE ESTIMATION and EDGE SAMPLING can be solved with high probability by using $\tilde{\Theta}\left(|V|/\sqrt{|E|}\right)$ LOCAL queries, where each local query is either a DEGREE or a NEIGHBOR or an ADJACENCY query [25, 21]. Thus, we observe that for INDUCED EDGE ESTIMATION and INDUCED EDGE SAMPLING, the ADJACENCY query is as good as the entire gamut of LOCAL queries and RANDOM EDGE query. On a different note, our results on BILINEAR FORM ESTIMATION and ALMOST UNIFORMLY SAMPLING using IP query generalize the above mentioned results on EDGE ESTIMATION and EDGE SAMPLING using local queries. Note that IP oracle is a natural query oracle for graphs where the unknown matrix is the adjacency matrix of a graph, and we will discuss that in Remark 3.8 that IP query on the adjacency matrix graphs is stronger than the local queries.

In Section 3.1, we prove Theorems 3.1 and 3.2 by reduction from a problem in communication complexity. In Section 3.2, we discuss the way in which INDUCED DEGREE query simulates local queries in any induced subgraph (see Remark 3.8). This will imply that the lower bound results in Theorems 3.1 and 3.2 can be overcome if we have an access to INDUCED DEGREE query to the whole graph (see Corollary 3.9). However, the implication is more general and will be discussed in Section 3.2.

3.1 Limitations of local and random edge queries

The proofs of the lower bounds use communication complexity. We will first provide a rudimentary introduction to communication complexity. For a detailed introduction to communication complexity refer to the following books [27, 31].

Brief introduction to communication complexity

In two-party communication complexity there are two parties, Alice and Bob, that wish to compute a function $\Pi : \{0, 1\}^N \times \{0, 1\}^N \rightarrow \{0, 1\}$. Alice is given $\mathbf{x} \in \{0, 1\}^N$ and Bob is given $\mathbf{y} \in \{0, 1\}^N$. Let x_i (y_i) denotes the i -th bit of \mathbf{x} (\mathbf{y}). While the parties know the function Π , Alice does not know \mathbf{y} , and similarly Bob does not know \mathbf{x} . Thus they communicate bits following a pre-decided protocol \mathcal{P} in order to compute $\Pi(\mathbf{x}, \mathbf{y})$. We say a randomized protocol \mathcal{P} computes Π if for all $(\mathbf{x}, \mathbf{y}) \in \{0, 1\}^N \times \{0, 1\}^N$ we have $\mathbb{P}[\mathcal{P}(\mathbf{x}, \mathbf{y}) = \Pi(\mathbf{x}, \mathbf{y})] \geq 2/3$. The model provides the parties access to a common random string of arbitrary length. The cost of the protocol \mathcal{P} is the maximum number of bits communicated, where maximum is over all inputs $(\mathbf{x}, \mathbf{y}) \in \{0, 1\}^N \times \{0, 1\}^N$. The communication complexity of the function is the cost of the most efficient protocol computing Π .

Proofs of Theorems 3.1 and 3.2

We will use the following problem in our lower bound proofs.

► **Definition 3.4** (k -INTERSECTION). Let $k, N \in \mathbb{N}$ such that $k \leq N$. Let $S = \{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \{0, 1\}^N, \sum_{i=1}^N x_i y_i = k \text{ or } 0\}$. The k -INTERSECTION function over N bits is a partial function denoted by k -INTERSECTION : $S \rightarrow \{0, 1\}$, and is defined as follows: k -INTERSECTION(\mathbf{x}, \mathbf{y}) = 1 if $\sum_{i=1}^N x_i y_i = k$ and 0, otherwise.

► **Lemma 3.5** ([27]). *Let $k, N \in \mathbb{N}$ such that $k \leq N$. The randomized communication complexity of k -INTERSECTION function on N bits is $\Omega(N/k)$.*

Proof of Theorem 3.1. We give a reduction from m_s -INTERSECTION problem over $N = s^2$ bits. Let $\mathbf{x} = (x_{ij}) \in \{0, 1\}^N$ be such that $i, j \in [s]$. Similarly, let $\mathbf{y} \in \{0, 1\}^N$. It is promised that Alice and Bob will be given \mathbf{x} and \mathbf{y} such that there are either 0 intersections or exactly m_s intersections, i.e., either $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ or m_s . Now we define a graph $G_{(\mathbf{x}, \mathbf{y})}(V(G), E(G))$ as follows where \sqcup denotes disjoint union.

- $|V(G)| = \Theta(s)$. $V(G) = S_A \sqcup S_B \sqcup T_A \sqcup T_B \sqcup C$ such that S_A, S_B, T_A, T_B are independent sets and $|S_A| = |S_B| = |T_A| = |T_B| = s$ and $|C| = \Theta(s)$. Note that $V(G)$ is independent of \mathbf{x} and \mathbf{y} ;
- The subgraph (of $G_{(\mathbf{x}, \mathbf{y})}$) induced by C is a fixed graph, independent of \mathbf{x} and \mathbf{y} , having exactly m_s edges. Also there are no edges in $G_{(\mathbf{x}, \mathbf{y})}$ between the vertices of C and $V(G) \setminus C$.
- The edges in the subgraph (of $G_{(\mathbf{x}, \mathbf{y})}$) induced by $V(G) \setminus C = S_A \sqcup T_A \sqcup S_B \sqcup T_B$ depend on \mathbf{x} and \mathbf{y} as follows. Let $S_A = \{s_i^A : i \in [s]\}$, $T_A = \{t_i^A : i \in [s]\}$, $S_B = \{s_i^B : i \in [s]\}$ and $T_B = \{t_i^B : i \in [s]\}$. For $i, j \in [s]$, if $x_{ij} = y_{ij} = 1$, then $(s_i^A, t_j^B) \in E(G)$ and $(s_i^B, t_j^A) \in E(G)$. For $i, j \in [s]$ if either $x_{ij} = 0$ or $y_{ij} = 0$, then $(s_i^A, t_j^A) \in E(G)$ and $(s_i^B, t_j^B) \in E(G)$;

The graph $G_{(\mathbf{x}, \mathbf{y})}$ can be uniquely generated from \mathbf{x} and \mathbf{y} . Moreover, Alice and Bob need to communicate to learn *useful* information about $G_{(\mathbf{x}, \mathbf{y})}$. Observation 3.6 follows from the construction that shows the relation between the number of edges in the subgraph induced by $S_A \sqcup T_B \sqcup C$ with $\langle \mathbf{x}, \mathbf{y} \rangle$, where $\mathbf{x}, \mathbf{y} \in \{0, 1\}^N$ are such that either $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ or $\langle \mathbf{x}, \mathbf{y} \rangle = m_s$.

► **Observation 3.6.** (i) $|S_A \sqcup T_B \sqcup C| = \Theta(s)$, (ii) irrespective of x and y : $|E_{S_A}| = |E_{S_B}| = |E_{T_A}| = |E_{T_B}| = 0$, also the degree of each vertex in $S_A \sqcup T_A \sqcup S_B \sqcup T_B$ is same (i.e., s), and (iii) if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$, then $|E_{S_A \sqcup T_B \sqcup C}| = m_s$, (iv) if $\langle \mathbf{x}, \mathbf{y} \rangle = m_s$, then $|E_{S_A \sqcup T_B \sqcup C}| = 2m_s$.

The following observation completes the proof of the theorem.

► **Observation 3.7.** Alice and Bob can deterministically determine answer for each local query to graph $G_{(\mathbf{x}, \mathbf{y})}$ by communicating $\mathcal{O}(1)$ bits.

We will give the proof of the above observation at the end of this subsection. ◀

Proof of Theorem 3.2. For clarity, we prove the theorem for $\varepsilon = 1/4$. However, the proof can be extended for any $\varepsilon \in (0, 1/2)$. We use the same set up and construction as in Theorem 3.1 with $S = S_A \sqcup S_B \sqcup C$. Let \mathcal{A} be an algorithm that almost uniformly samples edges from the induced graph $G_S = (S, E_S)$ making T queries, with probability $99/100$. Using \mathcal{A} we give another algorithm \mathcal{A}' that decides whether $|E_S| = m_s$ or $|E_S| = 2m_s$ by using $\mathcal{O}(T)$ queries, with probability at least $2/3$. From the reduction presented in Theorem 3.1, Alice and Bob can use \mathcal{A}' to solve m_s -INTERSECTION over $N = s^2$ bits, and hence $T = \Omega(\frac{s^2}{m_s})$.

\mathcal{A}' runs \mathcal{A} 10 times independently to obtain edges e_1, \dots, e_{10} . Note that each edge e_i is sampled almost uniformly. If at least one e_i satisfies $e_i \in E_{S_A \sqcup T_B}$, then \mathcal{A}' reports that $|E_S| = 2m_s$. Otherwise, \mathcal{A}' reports that $|E_S| = m_s$. The query cost of \mathcal{A}' is $\Theta(T)$.

If $|E_S| = m_s$, then there is no edge in the subgraph induced by $S_A \sqcup T_B$. So, in this case, all the edges reported by \mathcal{A}' are from the subgraph induced by C . Now consider when $|E_S| = 2m_s$. In this case, the subgraph induced by $S_A \sqcup T_B$ and C have exactly m_s edges each. So, by the assumption of the algorithm \mathcal{A} , the probability that any particular e_i is present in the subgraph induced by $S_A \sqcup T_B$ is at least $1/2 - \varepsilon = 1/4$ (since we are

analyzing for $\varepsilon = 1/4$). So, under the conditional space that all the ten runs of \mathcal{A} succeed, the probability that none of the ten edges sampled by \mathcal{A} is from the subgraph induced by $S_A \sqcup T_B$ is at most $(1 - 1/4)^{10} < 1/10$. As each run of algorithm \mathcal{A} succeeds with probability at least $99/100$, all the ten runs of the algorithm \mathcal{A} succeeds with probability at least $9/10$. So, the probability that algorithm \mathcal{A}' succeeds is at least $9/10 \cdot (1 - 1/10) > 2/3$. ◀

We will finish this subsection with the proof of Observation 3.7 used in the proof of Theorem 3.1.

Proof of Observation 3.7.

DEGREE query: By Observation 3.6 (ii), the degree of every vertex in $V(G) \setminus C$ is s . Also, the subgraph induced by C is a fixed graph disconnected from the rest. That is Alice and Bob know the degree of every vertex in C . Therefore, any DEGREE query can be simulated without any communication.

NEIGHBOR query: Observe that Alice and Bob can get the answer to any neighbor query involving a vertex in C without any communication. Now, consider the set S_A . The labels of the j neighbors of any vertex in $s_i^A \in S_A$ are as follows: for $j \in [s]$, the j -th neighbor of s_i^A is either t_j^B or s_j^A depending on whether $x_{ij} = y_{ij} = 1$ or not, respectively. So, any NEIGHBOR query involving vertex in S_A can be answered by 2 bits of communication. Similar arguments also hold for the vertices in $S_B \sqcup T_A \sqcup T_B$.

ADJACENCY query: Observe that each adjacency query can be answered by at most 2 bits of communication, and it can be argued like the NEIGHBOR query.

RANDOM EDGE query: By Observation 3.6 (ii), the degree of every vertex in $V(G) \setminus C$ is s irrespective of the inputs of Alice and Bob. Also, they know the entire subgraph induced by the vertex set C . Also, C is disconnected from the rest. Alice and Bob use shared randomness to sample a vertex in V proportional to its degree. Let $r \in V$ be the sampled vertex. They again use shared randomness to sample an integer j in $[d(v)]$ uniformly at random. Then they determine the j -th neighbor of r using NEIGHBOR query. Observe that this procedure simulates a RANDOM EDGE query by using at most 2 bits of communication. ◀

3.2 A query model for induced subgraphs

Observe that INDUCED DEGREE query can simulate any LOCAL queries on subgraphs.

► **Remark 3.8.** Let us have an INDUCED DEGREE query oracle access to an unknown graph $G(V, E)$. Consider any $X \subseteq V(G)$ and G_X , the subgraph of G induced by X . Then

- (i) Any query to G_X , which is either a DEGREE or ADJACENCY, can be answered by one INDUCED DEGREE query to G .
- (ii) Moreover, any NEIGHBOR query to G_X can be answered by $\mathcal{O}(\log |X|)$ INDUCED DEGREE query to G by *binary search*.

The above remark together with the edge estimation result of Goldreich and Ron [25], and edge sampling result of Eden and Rosenbaum [21], will give us the following result.

► **Corollary 3.9** (Estimating and sampling edges in induced subgraphs). *Let us assume that the query algorithms have access to INDUCED DEGREE query to an unknown graph $G = (V(G), E(G))$. There exists an algorithm that takes a subset $S \subseteq V(G)$ and $\varepsilon \in (0, 1)$ as inputs, and outputs a $(1 \pm \varepsilon)$ -approximation to $|E_S|$, with high probability, using $\tilde{\mathcal{O}}(|S|/\sqrt{E_S})$ INDUCED DEGREE queries to G . Also, there exists an algorithm that ε -almost uniformly samples edges in E_S , with high probability, using $\tilde{\mathcal{O}}(|S|/\sqrt{E_S})$ INDUCED DEGREE queries.*

► **Remark 3.10.** More generally, Remark 3.8 implies that any problem \mathcal{P} on a graph G that can be solved by using $f(|V(G)|, |E(G)|)$ local queries, can also be solved on any induced subgraph G_S , where $S \subseteq V(G)$, of G by using $f(|V(G_S)|, |E(G_S)|) \cdot \mathcal{O}(\log |V(G_S)|)$ INDUCED DEGREE queries.

4 Bilinear form estimating and sampling entries of a matrix

4.1 Algorithm for Bilinear Form Estimation

To give the main ideas behind the algorithm for BILINEAR FORM ESTIMATION, we will discuss, in this section, the algorithm for estimating $\mathbf{1}^T A \mathbf{1}$ using IP access to A , with A being symmetric. The algorithm for the special case is inspired by [25]. In the full version of the paper ([13]) we show how the algorithm for this special case can be extended for the general problem of estimating $\mathbf{x}^T A \mathbf{y}$, where $A \in [[\rho]]^{n \times n}$, $\mathbf{x} \in [[\gamma_1]]^n$ and $\mathbf{y} \in [[\gamma_2]]^n$. We will now give an outline of the proof of the following theorem.

► **Theorem 4.1.** *There exists a query algorithm for BFE that takes $\varepsilon \in (0, 1/2)$ as input and determines a $(1 \pm \varepsilon)$ -approximation to $\mathbf{1}^T A \mathbf{1}$ with high probability by using $\tilde{\mathcal{O}}\left(\frac{\sqrt{\rho n}}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$ IP queries to a symmetric matrix $A \in [[\rho]]^{n \times n}$. Moreover, the algorithm only uses IP query of the form $\langle A_{k*}, \mathbf{u} \rangle$ for some $k \in [n]$ and $\mathbf{u} \in \{0, 1\}^n$.*

The algorithms for BILINEAR FORM ESTIMATION and SAMPLE ALMOST UNIFORMLY (Section 4.2) will use a subroutine, which takes as input a given row $i \in [n]$ of A and a non-empty set $S \subseteq [n]$, and outputs A_{ij} , where $j \in S$, with probability $A_{ij} / \left(\sum_{j \in S} A_{ij}\right)$.

Algorithm 1 REGR(\mathbf{x}, i).

Input: A vector $\mathbf{x} \in \{0, 1\}^n$ such that the 1's in \mathbf{x} are consecutive and the number of 1's is a power of 2, an integer $i \in [n]$ and IP access to a matrix A .

Output: Ordered pair (i, j) with probability $\frac{A_{ij} \cdot x_j}{\langle A_{i*}, \mathbf{x} \rangle}$.

begin

if (the number of 1's in \mathbf{x} is 1) **then**

 | Report the ordered pair (ij^*) where $x_{j^*} = 1$.

end

else

 | Form a vector \mathbf{y} (\mathbf{z}) in $\{0, 1\}^n$ by setting second (first) half of the nonzero elements in \mathbf{x} to 0 and keeping the remaining elements unchanged.

 | Determine $\langle A_{i*}, \mathbf{y} \rangle$ and $\langle A_{i*}, \mathbf{z} \rangle$.

 | With probability $\frac{\langle A_{i*}, \mathbf{y} \rangle}{\langle A_{i*}, \mathbf{x} \rangle}$ report REGR(\mathbf{y}, i) and with probability $\frac{\langle A_{i*}, \mathbf{z} \rangle}{\langle A_{i*}, \mathbf{x} \rangle}$ report REGR(\mathbf{z}, i).

end

end

► **Observation 4.2.** There exists an algorithm REGR (See Algorithm 1) that takes $i \in [n]$ and $\mathbf{x} \in \{0, 1\}^n$ as inputs, outputs A_{ij} with probability $A_{ij} x_j / \left(\sum_{j \in [n]} A_{ij} x_j\right)$ by using $\mathcal{O}(\log n)$ IP queries to matrix A .

We will now discuss in the following paragraphs the details of the algorithm (Algorithm 2) for estimating $\mathbf{1}^T A \mathbf{1}$. The ingredients, to prove the correctness of Algorithm 2, are formally stated in Lemma 4.6. The approximation guarantee of Algorithm 2, which matches the guarantee mentioned in Theorem 4.1, is given in Claim 4.7.

Partition of rows of A induced by ε

Given ε as input, we argue that the rows of the symmetric matrix A can be partitioned into “buckets” such that the total number of buckets is small and every row in a particular bucket has approximately the same total weight. Consider a partition of $[n]$, that corresponds to the set of the indices of the rows of the symmetric matrix A , into buckets with the property that all j s present in a particular bucket B_i have approximately the same value of $\langle A_{j*}, \mathbf{1} \rangle$. Let $t = \lceil \log_{1+\beta}(\rho n) \rceil + 1$, where $\beta \leq \varepsilon/8$. For $i \in [t]$, we define the set $B_i := \{j \in [n] : (1 + \beta)^{i-1} \leq \langle A_{j*}, \mathbf{1} \rangle < (1 + \beta)^i\}$. Since $A_{ij} \leq \rho$, the maximum number of such buckets B_i required are at most $t = \lceil \log_{1+\beta}(\rho n) \rceil + 1$. Now consider the following fact that will be used in our analysis.

► **Fact 4.3.** For every $i \in [t]$, $(1 + \beta)^{i-1} |B_i| \leq \sum_{j \in B_i} \langle A_{j*}, \mathbf{1} \rangle < (1 + \beta)^i |B_i|$.

Based on the number of rows in a bucket, we classify the buckets to be either *large* or *small*. To define the large and small buckets, we require a lower bound ℓ on the value of $m = \mathbf{1}^T A \mathbf{1}$. Moreover, let us assume that, $m/6 \leq \ell \leq m$. However, this restriction can be removed by using standard techniques from property testing. For details, see [13, Section 4].

► **Definition 4.4.** We fix a threshold $\theta = \frac{1}{t} \cdot \frac{1}{n} \sqrt{\frac{\varepsilon}{8} \cdot \frac{\ell}{\rho}}$. For $i \in [t]$, we define the set B_i to be a *large bucket* if $|B_i| \geq \theta n$, otherwise B_i is defined to be a *small bucket*. Thus, the set of large buckets L is defined as $L = \{i \in [t] : |B_i| \geq \theta n\}$, and $[t] \setminus L$ is the set of small buckets.

Let $V, U \subseteq [n]$ be the sets of indices of rows that lie in large and small buckets, respectively. For $I \subseteq [n]$, let \mathbf{x}_I denote the sub-vector of \mathbf{x} induced by the indices present in I . Similarly, for $I, J \subseteq [n]$, let A_{IJ} denote the sub-matrix of A where the rows and columns are induced by the indices present in I and J , respectively. Observe that,

$$\mathbf{1}^T A \mathbf{1} = \mathbf{1}_V^T A_{VV} \mathbf{1}_V + \mathbf{1}_V^T A_{VU} \mathbf{1}_U + \mathbf{1}_U^T A_{UV} \mathbf{1}_V + \mathbf{1}_U^T A_{UU} \mathbf{1}_U.$$

2-Approximation of $\mathbf{1}^T A \mathbf{1}$

Note that at this point we know β and, upon querying $\langle A_{j*}, \mathbf{1} \rangle$, we can determine the bucket to which j belongs, for $j \in [n]$. The algorithm begins by sampling a subset S of rows of A , such that $|S| = K$, independently and uniformly at random with replacement, and for each sampled row j , the algorithm determines $\langle A_{j*}, \mathbf{1} \rangle$ by using IP oracle. This determines the bucket in which each sampled row belongs. Depending on the number of sampled rows present in different buckets, our algorithm classifies each bucket as either large or small. Let \tilde{V} and \tilde{U} be the indices of the rows present in large and small buckets, respectively. Note that the algorithm does not find \tilde{V} and \tilde{U} explicitly – these are used only for analysis purposes.

Observe that,

$$\mathbf{1}^T A \mathbf{1} = \mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{V}} \mathbf{1}_{\tilde{V}} + \mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{U}} \mathbf{1}_{\tilde{U}} + \mathbf{1}_{\tilde{U}}^T A_{\tilde{U}\tilde{V}} \mathbf{1}_{\tilde{V}} + \mathbf{1}_{\tilde{U}}^T A_{\tilde{U}\tilde{U}} \mathbf{1}_{\tilde{U}}.$$

We can show that $\mathbf{1}_{\tilde{U}}^T A_{\tilde{U}\tilde{U}} \mathbf{1}_{\tilde{U}}$ is at most $\frac{\varepsilon}{4} \ell$, where ℓ is a lower bound on $\mathbf{1}^T A \mathbf{1}$. Thus,

$$\mathbf{1}^T A \mathbf{1} \approx \mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{V}} \mathbf{1}_{\tilde{V}} + \mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{U}} \mathbf{1}_{\tilde{U}} + \mathbf{1}_{\tilde{U}}^T A_{\tilde{U}\tilde{V}} \mathbf{1}_{\tilde{V}}.$$

Lemma 4.6 shows that for a sufficiently large K , with high probability, the fraction of rows in any large bucket is approximately preserved in the sampled set of rows. Also observe that we know tight (upper and lower) bounds on $\langle A_{j*}, \mathbf{1} \rangle$ for every row j , where $j \in \tilde{V}$. Thus, the random sample of S rows, such that $|S| = K$, approximately preserves $\mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{V}} \mathbf{1}_{\tilde{V}} + \mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{U}} \mathbf{1}_{\tilde{U}}$. Observe that this is already 2-approximation of $\mathbf{1}^T A \mathbf{1}$.

Using REGR for tight approximation

In order to get a $(1 \pm \varepsilon)$ -approximation to $\mathbf{1}^T \mathbf{A} \mathbf{1}$, we need to estimate $\mathbf{1}_{\tilde{U}}^T A_{\tilde{U}\tilde{V}} \mathbf{1}_{\tilde{V}}$, which is same as estimating $\mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{U}} \mathbf{1}_{\tilde{U}}$ since A is a symmetric matrix. We estimate $\mathbf{1}_{\tilde{V}}^T A_{\tilde{V}\tilde{U}} \mathbf{1}_{\tilde{U}}$, that is, the sum of A_{ij} s such that $i \in \tilde{V}$ and $j \in \tilde{U}$, as follows. For each bucket B_i that is declared as large by the algorithm, we select *enough* number of rows randomly with replacement from $S_i = S \cap B_i$, invoke REGR for each selected row in S_i and increase the count by 1 if the element A_{ij} reported by REGR be such that $j \in \tilde{U}$. A formal description of our algorithm is given in Algorithm 2. Now, we focus on the correctness proof of our algorithm for BFE.

Algorithm 2 BFE (ℓ, ε).

Input: An estimate ℓ for $\mathbf{1}^T \mathbf{A} \mathbf{1}$ and $\varepsilon \in (0, 1/2)$.

Output: \hat{m} , which is a $(1 \pm \varepsilon)$ -approximation of $\mathbf{1}^T \mathbf{A} \mathbf{1}$.

begin

Independently select $K = \Theta\left(\frac{\sqrt{\rho n}}{\sqrt{\ell}} \cdot \varepsilon^{-4.5} \cdot \log^2(\rho n) \cdot \log(1/\varepsilon)\right)$ rows of A uniformly at random and let S denote the multiset of the selected indices (of rows) sampled. For $i \in [t]$, let $S_i = B_i \cap S$.

Let $\tilde{L} = \left\{i : \frac{|S_i|}{|S|} \geq \frac{1}{t} \cdot \frac{1}{n} \sqrt{\frac{\varepsilon}{6} \cdot \frac{\ell}{\rho}}\right\}$. Note that \tilde{L} is the set of buckets that the algorithm declares to be large. Similarly, $[t] \setminus \tilde{L}$ is the set of buckets declared to be small by the algorithm.

For every $i \in \tilde{L}$, select $|S_i|$ samples uniformly at random from S_i , with replacement, and let Z_i be the set of samples obtained. For each $z \in Z_i$, make a REGR($\mathbf{1}, z$) query and let $A_{zk_z} = \text{REGR}(\mathbf{1}, z)$. Let Y_z be a random variable that takes value 1 if $k_z \in \tilde{U}$ and 0, otherwise.

Determine $\tilde{\alpha}_i = \frac{\sum_{z \in Z_i} Y_z}{|S_i|}$.

Output $\hat{m} = \frac{n}{K} \sum_{i \in \tilde{L}} (1 + \tilde{\alpha}_i) \cdot |S_i| \cdot (1 + \beta)^i$.

end

To prove that \hat{m} is a $(1 \pm \varepsilon)$ -approximation of $m = \mathbf{1}^T \mathbf{A} \mathbf{1}$, we need the following definition and the technical Lemma 4.6.

► **Definition 4.5.** For $i \in L$, α_i is defined as $\frac{\sum_{u \in B_i} \langle A_{u*}, \mathbf{1}_{\tilde{U}} \rangle}{\sum_{u \in B_i} \langle A_{u*}, \mathbf{1} \rangle}$.

► **Lemma 4.6.** For a suitable choice of constant in $\Theta(\cdot)$ for selecting K samples in Algorithm 2, the followings hold with high probability:

- (i) For each $i \in L$, we have $\frac{|S_i|}{K} = (1 \pm \frac{\varepsilon}{4}) \frac{|B_i|}{n}$.
- (ii) For each $i \in [t] \setminus L$, $\frac{|S_i|}{K} < \frac{1}{t} \cdot \frac{1}{n} \sqrt{\frac{\varepsilon}{6} \cdot \frac{\ell}{\rho}}$.
- (iii) We have $|\tilde{U}| < \sqrt{\frac{\varepsilon}{4} \cdot \frac{\ell}{\rho}}$, where $\tilde{U} = \{j \in B_i : i \in [t] \setminus \tilde{L}\}$.
- (iv) For every $i \in \tilde{L}$, (a) if $\alpha_i \geq \frac{\varepsilon}{8}$, then $\tilde{\alpha}_i = (1 \pm \frac{\varepsilon}{4}) \alpha_i$, and (b) if $\alpha_i < \varepsilon/8$, then $\tilde{\alpha}_i < \varepsilon/4$.

The above Lemma can be proved by using Chernoff bound (see Appendix A). Now, we have all the ingredients to show the following claim, which shows that \hat{m} is a $(1 \pm \varepsilon)$ -approximation of $m = \mathbf{1}^T \mathbf{A} \mathbf{1}$. The following claim is proved in the full version of the paper [13].

▷ **Claim 4.7.** With high probability, we have,

- (i) $\widehat{m} \geq (1 - \frac{\varepsilon}{2}) (m - \frac{\varepsilon}{4} \ell)$, and
- (ii) $\widehat{m} \leq (1 + \frac{3\varepsilon}{4}) m$, where $m = \mathbf{1}^T A \mathbf{1}$.

Recall that we have assumed $m/6 \leq \ell \leq m$. Under this assumption, the above claim says that \widehat{m} is in fact a $(1 \pm \varepsilon)$ -approximation to m . From the description of Algorithm 2, the number of IP queries made by the algorithm is $\tilde{O}\left(\frac{\sqrt{\rho n}}{\sqrt{\ell}}\right) = \tilde{O}\left(\frac{\sqrt{\rho n}}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$ as $m/10 \leq \ell \leq m$. We will discuss how to remove the assumption, that $m/6 \leq \ell \leq m$, by using a standard technique in property testing (see the full version of the paper [13]). So, we are done with the proof of Theorem 4.1.

4.2 Algorithm for Sampling Almost Uniformly

In this section, we will be proving the following theorem on almost uniformly sampling the entries of a symmetric matrix $A \in [[\rho]]^{n \times n}$. The algorithm for the special case is inspired by [21]. In the full version of the paper ([13]), we show how this algorithm can be extended to solve the more general $\text{SAU}_A(\mathbf{x}, \mathbf{y})$ problem.

► **Theorem 4.8.** *Let $A \in [[\rho]]^{n \times n}$ be an unknown symmetric matrix with IP query access. There exists an algorithm that takes $\varepsilon \in (0, 1)$ as input and with high probability outputs a sample from a distribution on $[n] \times [n]$, such that each $(i, j) \in [n] \times [n]$ is sampled with probability p_{ij} satisfying:*

$$p_{ij} = (1 \pm \varepsilon) \frac{A_{ij}}{\left(\sum_{1 \leq i, j \leq n} A_{ij}\right)}.$$

Moreover, the algorithm makes $\tilde{O}\left(\frac{\sqrt{\rho n}}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$ IP queries to the matrix A of the form $\langle A_{k*}, \mathbf{u} \rangle$ for some $k \in [n]$ and $\mathbf{u} \in \{0, 1\}^n$.

Our algorithm for SAMPLE ALMOST UNIFORMLY is a generalization of Eden and Rosenbaum's algorithm for *sampling edges of an unweighted graph* [21]. First, consider the following strategy by which we sample each ordered pair $(i, j) \in [n] \times [n]$ proportional to A_{ij} when the matrix A is such that $\langle A_{i*}, \mathbf{1} \rangle$ is the same for each $i \in [n]$.

Strategy-1: Sample $r \in [n]$ uniformly at random and then sample an ordered pair of the form (r, j) from the r -th row using REGR query. Observe that this strategy fails when $\langle A_{i*}, \mathbf{1} \rangle$'s are not the same for every $i \in [n]$. So, the modified strategy is as follows.

Strategy-2: Sample $r \in [n]$ with probability $\frac{\langle A_{r*}, \mathbf{1} \rangle}{\mathbf{1}^T A \mathbf{1}}$ and then sample an ordered pair of the form (r, j) from the r -th row by using REGR query.

Note that Strategy-2 samples each ordered pair (i, j) proportional to A_{ij} . However, there are two challenges in executing Strategy-2:

- (i) We do not know the value of $\mathbf{1}^T A \mathbf{1}$.
- (ii) We need $\Omega(n)$ queries to determine $\langle A_{r*}, \mathbf{1} \rangle$ for each $r \in [n]$.

The first challenge can be taken care of by finding an estimate \widehat{m} for $\mathbf{1}^T A \mathbf{1}$, with high probability, by using Theorem 4.1 such that $\widehat{m} = \Theta(\mathbf{1}^T A \mathbf{1})$. To cope up with the second challenge, we partition the elements as well as rows into two classes as defined in Definition 4.9. In what follows, we consider a parameter τ in terms of which we base our discussion as well as algorithm. τ is a function of \widehat{m} that will evolve over the calculation and will be $\tau = \sqrt{\frac{\widehat{\rho m}}{\varepsilon}}$.

► **Definition 4.9.** The i -th row of the matrix is *light* if $\langle A_{i,*}, \mathbf{1} \rangle \leq \tau$, otherwise i -th row is *heavy*. Any order pair (i, j) , for a fixed i , is light (heavy) if the i -th row is light (heavy).

We denote the set of all light (heavy) ordered pairs by \mathcal{L} (\mathcal{H}). Also, let $I(L)$ ($I(H)$) denote the set of light (heavy) rows of the matrix A . Let $w(\mathcal{L}) = \sum_{A_{ij} \in \mathcal{L}} A_{ij}$ and $w(\mathcal{H}) = \sum_{A_{ij} \in \mathcal{H}} A_{ij}$.

Our algorithm consists of repeated invocation of two subroutines, that is, SAMPLE-LIGHT and SAMPLE-HEAVY. Both SAMPLE-LIGHT and SAMPLE-HEAVY succeed with *good* probability and sample elements from \mathcal{L} and \mathcal{H} almost uniformly, respectively. The threshold τ is set in such a way that there are *large*⁸ number of light rows and small number of heavy rows. In SAMPLE-LIGHT, we select a row uniformly at random, and if the selected row is light, then we sample an ordered pair from the selected row randomly using REGR. This gives us an element from \mathcal{L} uniformly. However, the same technique will not work for SAMPLE-HEAVY as we have few heavy rows. To cope up with this problem, we take a row uniformly at random and if the selected row is light, we sample an ordered pair from the selected row randomly using REGR. Let (i, j) be the output of the REGR query. Then we go to the j -th row, if it is heavy, and then select an ordered pair from the j -th row randomly using REGR query.

The formal algorithms for SAMPLE-LIGHT and SAMPLE-HEAVY are given in Algorithm 3 and Algorithm 4, respectively. The formal correctness of SAMPLE-LIGHT and SAMPLE-HEAVY are given in Lemmas 4.10 and 4.11, respectively. We give the final algorithm along with its proof of correctness in Theorem 4.8.

■ **Algorithm 3** SAMPLE-LIGHT.

Input: An estimate \widehat{m} for $\mathbf{1}^T A \mathbf{1}$ and a threshold τ .

Output: $(i, j) \in \mathcal{L}$ with probability $\frac{A_{ij}}{n\tau}$.

begin

 Select a row $r \in [n]$ uniformly at random.

if ($r \in I(\mathcal{L})$, that is, $\langle A_{r,*}, \mathbf{1} \rangle$ is at most τ) **then**

 Return FAIL with probability $p = \frac{\tau - \langle A_{r,*}, \mathbf{1} \rangle}{\tau}$, and Return REGR($r, \mathbf{1}$) with probability $1 - p$ as the output.

end

 Return FAIL

end

► **Lemma 4.10.** SAMPLE-LIGHT succeeds with probability $\frac{w(\mathcal{L})}{n\tau}$. Let Z_ℓ be the output in case it succeeds. Then $\mathbb{P}(Z_\ell = (i, j)) = \frac{A_{ij}}{n\tau}$ if $(i, j) \in \mathcal{L}$, and $\mathbb{P}(Z_\ell = (i, j)) = 0$, otherwise. Moreover, SAMPLE-LIGHT makes $\mathcal{O}(\log n)$ queries.

► **Lemma 4.11.** SAMPLE-HEAVY succeeds with probability at most $\frac{w(\mathcal{H})}{n\tau}$ and at least $\left(1 - \frac{\widehat{m}}{\tau^2}\right) \frac{w(\mathcal{H})}{n\tau}$. Let Z_h be the output in case it succeeds. Then, $\left(1 - \frac{\widehat{m}}{\tau^2}\right) \frac{A_{ij}}{n\tau} \leq \mathbb{P}(Z_h = (i, j)) \leq \frac{A_{ij}}{n\tau}$ for each $(i, j) \in \mathcal{H}$, and $\mathbb{P}(Z_h = (i, j)) = 0$, otherwise. Moreover, SAMPLE-HEAVY makes $\mathcal{O}(\log n)$ queries.

Now we will use the above lemmas to prove Theorem 4.8.

⁸ Large is parameterized by τ .

■ **Algorithm 4** SAMPLE-HEAVY (\widehat{m}).

Input: An estimate \widehat{m} for $\mathbf{1}^T A \mathbf{1}$ and a threshold τ .
Output: $A_{ij} \in \mathcal{H}$ with probability at most $\frac{A_{ij}}{n\tau}$ and at least $\left(1 - \frac{\widehat{\rho m}}{\tau^2}\right) \frac{A_{ij}}{n\tau}$.

begin

- Select a row $r \in [n]$ uniformly at random;
- if** ($r \in I(\mathcal{L})$, that is, $\langle A_{r*}, \mathbf{1} \rangle$ is at most τ) **then**
 - Return FAIL with probability $p = \frac{\tau - \langle A_{r*}, \mathbf{1} \rangle}{\tau}$, and with probability $1 - p$ do the following;
 - $A_{rs} = \text{REGR}(r, \mathbf{1})$
 - If $s \in I(\mathcal{H})$, that is, $\langle A_{s*}, \mathbf{1} \rangle > \tau$, then Return $\text{REGR}(s, \mathbf{1})$ as the output.
 - Otherwise, Return FAIL;
- end**
- Return FAIL;

end

Proof of Theorem 4.8. Our algorithm first finds a rough estimate \widehat{m} for $\mathbf{1}^T A \mathbf{1}$, with high probability, by using Theorem 4.1 such that $\widehat{m} = \Theta(\mathbf{1}^T A \mathbf{1})$. For the rest of the proof, we work on the conditional probability space that $\widehat{m} = \Theta(\mathbf{1}^T A \mathbf{1})$. We set $\tau = \sqrt{\frac{\widehat{\rho m}}{\varepsilon}}$ and do the following for Γ times, where Γ is a parameter to be set later. With probability $1/2$, we invoke SAMPLE-LIGHT and with probability $1/2$, we invoke SAMPLE-HEAVY. If the ordered pair (i, j) is reported as the output by either SAMPLE-LIGHT or SAMPLE-HEAVY, we report that. If we get FAIL in all the trials, we report FAIL.

Now, let us consider a particular trial and compute the probability of success $\mathbb{P}(S)$, which is $\mathbb{P}(S) = \frac{1}{2} (\mathbb{P}(\text{SAMPLE-LIGHT succeeds}) + \mathbb{P}(\text{SAMPLE-HEAVY succeeds}))$. Observe that from Lemmas 4.10 and 4.11, we have,

$$\frac{1}{2} \left(\frac{w(\mathcal{L})}{n\tau} + \left(1 - \frac{\widehat{\rho m}}{\tau^2}\right) \frac{w(\mathcal{H})}{n\tau} \right) \leq \mathbb{P}(S) \leq \frac{1}{2} \left(\frac{w(\mathcal{L})}{n\tau} + \frac{w(\mathcal{H})}{n\tau} \right).$$

This implies $(1 - \varepsilon) \frac{\mathbf{1}^T A \mathbf{1}}{2n\tau} \leq \mathbb{P}(S) \leq \frac{\mathbf{1}^T A \mathbf{1}}{2n\tau}$ as $\tau = \sqrt{\frac{\widehat{\rho m}}{\varepsilon}}$ and using $w(\mathcal{L}) + w(\mathcal{H}) = \mathbf{1}^T A \mathbf{1}$.

Now, let us compute the probability of the event \mathcal{E}_{ij} , that is, the algorithm succeeds and it returns A_{ij} . If $A_{ij} \in \mathcal{L}$, by Lemma 4.10, we have $\mathbb{P}(Z = (i, j)) = \frac{1}{2} \cdot \frac{A_{ij}}{n\tau}$. Also, if $A_{ij} \in \mathcal{H}$, by Lemma 4.11, we have, $\left(1 - \frac{\widehat{\rho m}}{\tau^2}\right) \frac{A_{ij}}{2n\tau} \leq \mathbb{P}(Z = (i, j)) \leq \frac{A_{ij}}{2n\tau}$. So, for any (i, j) , we get $(1 - \varepsilon) \frac{A_{ij}}{2n\tau} \leq \mathbb{P}(\mathcal{E}_{ij}) \leq \frac{A_{ij}}{2n\tau}$. Let us compute the probability of \mathcal{E}_{ij} on the conditional probability space that the algorithm succeeds, that is, $\mathbb{P}(Z = (i, j) \mid S) = \frac{\mathbb{P}(\mathcal{E}_{ij})}{\mathbb{P}(S)}$, which lies in the interval $\left[(1 - \varepsilon) \frac{A_{ij}}{\mathbf{1}^T A \mathbf{1}}, (1 + \varepsilon) \frac{A_{ij}}{\mathbf{1}^T A \mathbf{1}} \right]$ as $\varepsilon \in (0, \frac{1}{2})$.

To boost the probability of success, we set $\Gamma = \mathcal{O}\left(\frac{n\sqrt{\rho}}{(1-\varepsilon)\sqrt{\varepsilon\widehat{m}}} \log n\right)$ for a suitable *large* constant in $\mathcal{O}(\cdot)$ notation. The query complexity of each call to SAMPLE-LIGHT and SAMPLE-HEAVY is $\mathcal{O}(\log n)$. Also note that our algorithm for SAMPLE ALMOST UNIFORMLY makes at most $\mathcal{O}\left(\frac{n\sqrt{\rho}}{(1-\varepsilon)\sqrt{\varepsilon\widehat{m}}} \log n\right)$ invocations to SAMPLE-LIGHT and SAMPLE-HEAVY. Hence, the total query complexity of our algorithm is $\widetilde{\mathcal{O}}\left(\frac{\sqrt{\rho n}}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$. ◀

■ **Table 1** Comparing IP with matrix-vector and vector-matrix-vector queries. Recall that MV and VMV stand for matrix-vector and vector-matrix-vector queries.

Problem	IP Query	VMV Query [32]	MV Query [40]
SYMMETRIC MATRIX	$\tilde{\Theta}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
DIAGONAL MATRIX	$\Theta(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
TRACE	$\Theta(n)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$
PERMUTATION MATRIX	$\Theta(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
DOUBLY STOCHASTIC MATRIX	$\Theta(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
IDENTICAL COLUMNS ⁹	$\tilde{\Theta}(n)$	$\tilde{\Theta}(n)$	$\tilde{\Theta}(n)$
ALL ONES COLUMNS	$\Theta(n)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$

5 Conclusion and discussions

Other matrix problems

Recently, vector-matrix query [40] and vector-matrix-vector query [32] were introduced to study a bunch of matrix, graph and statistics problems. As noted earlier, IP query oracle is in the same linear algebraic framework of vector-matrix (VM) query and vector-matrix-vector (VMV) query, but these queries are stronger than IP query. Study of the various matrix, graph and statistics problems, introduced in [40, 32], using IP query will be of independent interest. As a first step in that direction, in the full version of this paper [13], we study the query complexity of the following problems using IP queries.

- SYMMETRIC MATRIX: Is $A \in \{0, 1\}^{n \times n}$ a symmetric matrix?
- DIAGONAL MATRIX: Is A a diagonal matrix?
- TRACE: Compute the trace of the matrix A .
- PERMUTATION MATRIX: Is $A \in \{0, 1\}^{n \times n}$ a permutation matrix?
- DOUBLY STOCHASTIC MATRIX: Is $A \in \{0, 1\}^{n \times n}$ a doubly stochastic matrix?
- IDENTICAL COLUMNS: Does there exist two columns in $A \in \{0, 1\}^{n \times n}$ that are identical?
- ALL ONES COLUMN: Does there exist a column in A all of whose entries are 1?

Table 1 compares the query complexities of IP oracle with matrix-vector (MV) and vector-matrix-vector (VMV) queries.

Data structure complexity and open problems

Besides property testing, there have been extensive work concerning vector-matrix-vector product in data structure complexity and other models of computation like the cell probe model [15, 16, 17, 28, 30]. For the purposes of this paper, it is an interesting question to find a pre-processing scheme for the matrix such the IP queries on the matrix can be answered efficiently.

⁹ Upper and lower bounds results for IDENTICAL COLUMNS problem in [32, 40] uses vectors from $\{0, 1\}^n$ in their respective queries.

One of the open problems that is left from our work is to design algorithm and/or prove lower bound for BILINEAR FORM ESTIMATION and SAMPLING when the entries of the matrices are not necessarily positive. Here we would like to state that our technique does not work for a matrix with both positive and negative entries. Some other natural open questions are:

- Are there some special kind of matrices where we can solve BILINEAR FORM ESTIMATION and SAMPLING using fewer queries?
- Can we solve some other linear algebraic problems using IP queries?
- Are there other graph problems, where INDUCED DEGREE outperforms LOCAL queries?

References

- 1 Intel C++ Compiler Classic Developer Guide and Reference: Floating Point Dot Product Intrinsics. <https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/intrinsics/intrinsics-for-intel-streaming-simd-extensions-4-intel-sse4/vectorizing-compiler-and-media-accelerators/floating-point-dot-product-intrinsics.html>.
- 2 NVIDIA Developer: Cg 3.1 Toolkit Documentation. <https://developer.download.nvidia.com/cg/dot.html>.
- 3 K. J. Ahn, S. Guha, and A. McGregor. Analyzing Graph Structure via Linear Measurements. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 459–467, 2012.
- 4 M. Aliakbarpour, A. S. Biswas, T. Gouleakis, J. Peebles, R. Rubinfeld, and A. Yodpinyanee. Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling. *Algorithmica*, 80(2):668–697, 2018.
- 5 S. Assadi, D. Chakrabarty, and S. Khanna. Graph connectivity and single element recovery via linear and OR queries. In *European Symposium on Algorithms, ESA*, volume 204, pages 7:1–7:19, 2021.
- 6 S. Assadi, M. Kapralov, and S. Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 6:1–6:20, 2019.
- 7 M.-F. Balcan, Y. Li, D. P. Woodruff, and H. Zhang. Testing Matrix Rank, Optimally. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 727–746, 2019.
- 8 P. Beame, S. Har-Peled, S. N. Ramamoorthy, C. Rashtchian, and M. Sinha. Edge Estimation with Independent Set Oracles. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 38:1–38:21, 2018.
- 9 I. Ben-Eliezer, T. Kaufman, M. Krivelevich, and D. Ron. Comparing the strength of query types in property testing: the case of testing k -colorability. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1213–1222, 2008.
- 10 M. Benzi and C. Klymko. Total Communicability as a Centrality Measure. *Journal of Complex Networks*, 1:124–149, 2013.
- 11 A. Bishnu, A. Ghosh, S. Kolay, G. Mishra, and S. Saurabh. Parameterized Query Complexity of Hitting Set Using Stability of Sunflowers. In *International Symposium on Algorithms and Computation, ISAAC*, pages 25:1–25:12, 2018.
- 12 A. Bishnu, A. Ghosh, and G. Mishra. Distance Estimation Between Unknown Matrices Using Sublinear Projections on Hamming Cube. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 207, pages 44:1–44:22, 2021.
- 13 A. Bishnu, A. Ghosh, G. Mishra, and M. Paraashar. Efficiently sampling and estimating from substructures using linear algebraic queries. *CoRR*, abs/1906.07398, 2019. Version 2. [arXiv:1906.07398v2](https://arxiv.org/abs/1906.07398v2).

- 14 F. Bonchi, P. Esfandiar, D. F. Gleich, C. Greif, and L. V.S. Lakshmanan. Fast Matrix Computations for Pairwise and Columnwise Commute Times and Katz Scores. *Internet Mathematics*, 1-2(8):73–112, 2012.
- 15 D. Chakraborty, L. Kamma, and K. G. Larsen. Tight Cell Probe Bounds for Succinct Boolean Matrix-Vector Multiplication. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1297–1306, 2018.
- 16 A. Chattopadhyay, M. Koucký, B. Loff, and S. Mukhopadhyay. Simulation Beats Richness: New Data-Structure Lower Bounds. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1013–1020, 2018.
- 17 H. Dell, J. Lapinskas, and K. Meeks. Approximately Counting and Sampling Small Witnesses using a Colourful Decision Oracle. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2201–2211, 2020.
- 18 D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 1st edition, 2009.
- 19 T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately Counting Triangles in Sublinear Time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.
- 20 T. Eden, D. Ron, and C. Seshadhri. On Approximating the Number of k -Cliques in Sublinear Time. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 722–734, 2018.
- 21 T. Eden and W. Rosenbaum. On Sampling Edges Almost Uniformly. In *Symposium on Simplicity in Algorithms, SOSA*, pages 7:1–7:9, 2018.
- 22 E. Estrada and D. J. Higham. Network Properties Revealed through Matrix Functions. *SIAM Review*, 52:696–714, 2010.
- 23 U. Feige. On Sums of Independent Random Variables with Unbounded Variance and Estimating the Average Degree in a Graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.
- 24 O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.
- 25 O. Goldreich and D. Ron. Approximating Average Parameters of Graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.
- 26 J. L. Hennessy and D. A. Patterson. *Computer Architecture – A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.
- 27 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
- 28 K. G. Larsen and R. R. Williams. Faster Online Matrix-Vector Multiplication. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2182–2189, 2017.
- 29 N. Nisan. The demand query model for bipartite matching. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 592–599, 2021.
- 30 S. N. Ramamoorthy and C. Rashtchian. Equivalence of Systematic Linear Data Structures and Matrix Rigidity. In *Innovations in Theoretical Computer Science Conference, ITCS*, volume 151, pages 35:1–35:20, 2020.
- 31 A. Rao and A. Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.
- 32 C. Rashtchian, D. P. Woodruff, and H. Zhu. Vector-Matrix-Vector Queries for Solving Linear Algebra, Statistics, and Graph Problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 176, pages 26:1–26:20, 2020.
- 33 D. Ron and G. Tsur. The Power of an Example: Hidden Set Size Approximation Using Group Queries and Conditional Sampling. *ACM Trans. Comput. Theory*, 8(4):15:1–15:19, 2016.
- 34 A. Rubinfeld, T. Schramm, and S. M. Weinberg. Computing Exact Minimum Cuts Without Knowing the Graph. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 39:1–39:16, 2018.
- 35 J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, Upper Saddle River, NJ, 2010.
- 36 X. Shi and D. P. Woodruff. Sublinear Time Numerical Linear Algebra for Structured Matrices. In *AAAI Conference on Artificial Intelligence, AAAI*, pages 727–746, 2019.

- 37 L. J. Stockmeyer. The Complexity of Approximate Counting (Preliminary Version). In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 118–126, 1983.
- 38 L. J. Stockmeyer. On Approximation Algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985.
- 39 X. Sun, D. P. Woodruff, G. Yang, and J. Zhang. Querying a Matrix Through Matrix-Vector Products. In *International Colloquium on Automata, Languages, and Programming, ICALP*, pages 94:1–94:16, 2019.
- 40 X. Sun, D. P. Woodruff, G. Yang, and J. Zhang. Querying a Matrix Through Matrix-Vector Products. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 132, pages 94:1–94:16, 2019.

A Useful probability bounds

► **Lemma A.1** (See [18]). Let $X = \sum_{i=1}^n X_i$ where $X_i, i \in [n]$, are independent random variables, $X_i \in [0, 1]$ and $\mathbb{E}[X]$ is the expected value of X .

- (i) For $\varepsilon > 0$, we have $\Pr[|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}[X]] \leq \exp(-\varepsilon^2 \mathbb{E}[X]/3)$.
- (ii) Suppose $\mu_L \leq \mathbb{E}[X] \leq \mu_H$. Then, for all $0 < \varepsilon < 1$, we have
 - (a) $\Pr[X > (1 + \varepsilon)\mu_H] \leq \exp(-\varepsilon^2 \mu_H/3)$.
 - (b) $\Pr[X < (1 - \varepsilon)\mu_L] \leq \exp(-\varepsilon^2 \mu_L/2)$.

Derandomization via Symmetric Polytopes: Poly-Time Factorization of Certain Sparse Polynomials

Pranav Bisht   

Department of Computer Science & Engineering, IIT Kanpur, India

Nitin Saxena   

Department of Computer Science & Engineering, IIT Kanpur, India

Abstract

More than three decades ago, after a series of results, Kaltofen and Trager (J. Symb. Comput. 1990) designed a randomized polynomial time algorithm for factorization of multivariate circuits. Derandomizing this algorithm, even for restricted circuit classes, is an important open problem. In particular, the case of s -sparse polynomials, having individual degree $d = O(1)$, is very well-studied (Shpilka, Volkovich ICALP'10; Volkovich RANDOM'17; Bhargava, Saraf and Volkovich FOCS'18, JACM'20). We give a complete derandomization for this class assuming that the input is a *symmetric* polynomial over rationals. Generally, we prove an $s^{\text{poly}(d)}$ -sparsity bound for the factors of symmetric polynomials over any field. This characterizes the known worst-case examples of sparsity blow-up for sparse polynomial factoring.

To factor f , we use techniques from convex geometry and exploit symmetry (only) in the Newton polytope of f . We prove a crucial result about convex polytopes, by introducing the concept of “low min-entropy”, which might also be of independent interest.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Pseudorandomness and derandomization; Mathematics of computing → Combinatoric problems

Keywords and phrases Multivariate polynomial factorization, derandomization, sparse polynomials, symmetric polynomials, factor-sparsity, convex polytopes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.9

Related Version *Full Version*:

<https://www.cse.iitk.ac.in/users/nitin/papers/symmetricSparse.pdf>

Funding *Nitin Saxena*: DST-SERB (CRG/2020/000045) and N. Rama Rao Chair.

Acknowledgements Pranav thanks Ilya Volkovich and Vishwas Bhargava for useful discussions. The authors would also like to thank the anonymous reviewers for their useful comments that improved the presentation of results.

1 Introduction

Polynomial factorization is one of the most fundamental and central problems studied in computational algebra. In this paper, we concern ourselves with the problem of *multivariate* polynomial factorization which, given a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ over some field \mathbb{F} as an input, asks to compute all the irreducible factors of f . In a famous line of work, it was shown that this problem admits a polynomial time randomized algorithm [18, 20]. Finding an efficient *deterministic* algorithm continues to be a challenging open problem¹.

¹ We only want an efficient deterministic algorithm to reduce multivariate circuit (or even sparse) factorization to univariate factorization.



9:2 Sparse Polynomial Factorization

See [19, 38, 39, 35] for a detailed exposition. Polynomial factorization also has interesting connections with other problems such as circuit lower bounds [15], list decoding [34, 13] and cryptography [6].

The representation of the input polynomial is significant in the complexity of this problem. The randomized algorithms of [18, 20] are efficient when both the input polynomial and output factors are represented as general algebraic circuits (see Appendix B for definition). It is imperative to ask if derandomizing polynomial factorization will be easier for restricted circuit classes. One may believe the problem to be easier since the input polynomial is weak as it belongs to a restricted class. However, note that the demand is correspondingly stronger; we want output factors also to be given in this restricted class. A very natural interesting class of algebraic circuits, which is considered for this problem, is that of *sparse* polynomials. Sparsity of a polynomial f is defined as number of monomials in f with non-zero coefficients. Sparse polynomials can also be seen as *depth-2* ($\Sigma\Pi$) algebraic circuits. If we give sparse polynomial as input to Kaltofen's factorization algorithm, the output will be in the form of general algebraic circuit, but in *sparse polynomial factorization*, we want output factors also to be in sparse representation.

There is another motivation for studying sparse polynomials for the factorization problem, which arises from another fundamental problem called the *Polynomial identity testing* (PIT). Given an input polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ as an algebraic circuit, PIT asks to determine if f is identically zero. This problem also admits a simple and efficient randomized algorithm due to PIT Lemma [29, 42, 10, 26] and like factoring, finding a deterministic efficient algorithm is still open. It is easy to show that PIT reduces to factorization². In fact, [22] showed that the problem of derandomizing multivariate polynomial factoring is equivalent to derandomizing PIT, for general algebraic circuits. Showing this equivalence for other interesting circuit classes, was left as an open problem by [22]. The classes, which admit an efficient deterministic PIT algorithm are particularly interesting because if we show the equivalence for these classes, we also derandomize factoring for them. Sparse polynomials is one such natural class, for which we do have poly-time deterministic PIT algorithms [21].

Note that the run-time of sparse factorization algorithm will be lower bounded by the sparsity of factors, just for writing the output alone. In general, the sparse model is not closed under factors, i.e. sparse polynomials can have dense factors as shown in Examples 4.8 and 4.9. Therefore, the bounded *individual degree* setting is studied for sparse polynomial factorization [31, 36, 37, 2]. Individual degree of a polynomial is defined as the maximum degree of some variable appearing in it (See Section 2). This raises the following important question:

► **Question 1.1.** *Let f be an s -sparse polynomial with constant individual degree. Can f be deterministically factored in $\text{poly}(s)$ -time?*

We make progress towards a positive answer to this question by giving a $\text{poly}(s)$ -time deterministic factoring algorithm, when the sparse polynomial f is also symmetric. Symmetric polynomials are ones that remain the same even if any of the variables are interchanged (See Section 2 for formal definition). To the best of our knowledge, symmetry was not exploited before our work and the previous best deterministic algorithm for this class was a super-polynomial time algorithm by [2] with run-time $s^{O(\log n)}$, for general s -sparse polynomials.

² Observe that the polynomial $x^2 + y \cdot f$ factorizes if and only if $f = 0$.

The work of [2] partially derandomizes the factoring question for sparse polynomials having constant individual degree. The crucial derandomization step in their algorithm is proving an efficient sparsity bound for factors of the input polynomial f , which dominates the run-time of their algorithm. They were able to prove an $s^{O(\log n)}$ sparsity bound for factors of f . Our main contribution is to prove factor-sparsity bound of $s^{O(1)}$, when f is symmetric. The key combinatorial object associated with derandomization of sparse factoring or equivalently factor-sparsity, is the *Newton polytope* of f . We prove our bound by giving a completely new line of attack on the symmetric Newton polytope of f .

Symmetric polynomials are studied extensively both in computer science and mathematics. It is intriguing to explore the effect of symmetry in the polynomial factorization problem. Many multivariate polynomials $f \in \mathbb{F}[x_1, \dots, x_n]$ are constructed by “boosting” a univariate polynomial $a(X)$ by product or addition as shown below:

$$f = \prod_{i=1}^n a(x_i) \quad \text{or} \quad f = \left(\sum_{i=1}^n a(x_i) \right)^d \quad \text{for some } d \geq 1.$$

Some of the famous polynomials that are known to have dense factors (Example 4.8 and Example 4.9) are of this type. Such polynomials are actually symmetric by construction and are exactly captured by the results of this work. Therefore, the symmetric setting is important and non-trivial. Moreover, our proof techniques do *not* require symmetry in the coefficients of f , but merely in the support of f . Besides symmetric polynomials, our techniques also give polynomial-time deterministic factoring algorithm for another class of polynomials, which we call *low min-entropy* polynomials, that we define later.

1.1 Our results

Given a polynomial f , the *complete factorization* of f is a representation of f as $f_1^{e_1} f_2^{e_2} \dots f_k^{e_k}$, where f_1, \dots, f_k are co-prime, irreducible polynomials and e_1, \dots, e_k are positive integers. This representation is unique up to a reordering of f_i 's. Let \mathbb{Q} denote the field of rational numbers. We get a poly-time deterministic reduction from multivariate to univariate factoring over any field \mathbb{F} . In particular, this gives us a complete poly-time factoring algorithm over \mathbb{Q} below, by using the univariate factoring algorithm of [23].

► **Theorem 1.2** (Symmetric factoring algorithm). *Let $f \in \mathbb{Q}[x_1, \dots, x_n]$ be an s -sparse, symmetric polynomial with constant individual degree d . Let t be the maximum bit-complexity of the coefficients of f . Then, there is a deterministic algorithm that computes the complete factorization of f in at most $\text{poly}(s, n, t)$ -time.*

► **Remark.**

1. The exact complexity is $\text{poly}(s^{d^7 \log d} \cdot n^{d^2} \cdot t)$ -time.
2. For a finite field $\mathbb{F} = \mathbb{F}_{p^\ell}$, we get a deterministic $\text{poly}(s^{d^7 \log d} \cdot n^{d^2} \cdot \ell \log p)$ -time reduction to univariate factoring.
3. Throughout this paper (specifically in Theorems 1.2, 1.3 & 4.6), we get the same results if we replace symmetric polynomials with a more general class of polynomials, which we call *symmetric-support* polynomials. Let $\text{supp}(f)$ denote the set of monomials in f with non-zero coefficients. We call $f \in \mathbb{F}[x_1, \dots, x_n]$ a symmetric-support polynomial if for each monomial $x_1^{e_1} \dots x_n^{e_n} \in \text{supp}(f)$, we also have that $x_1^{e_{\sigma(1)}} \dots x_n^{e_{\sigma(n)}} \in \text{supp}(f)$ for every permutation $\sigma \in S_n$. For eg., $f = x_1^2 x_2 x_3 + 2x_1 x_2^2 x_3 - x_1 x_2 x_3^2$ is a symmetric-support polynomial that is not symmetric. While $f = x_1^2 x_2 x_3 + x_1 x_2^2 x_3$ is not symmetric-support.

9:4 Sparse Polynomial Factorization

The factoring algorithm in Theorem 1.2 is a direct consequence of our new sparsity bound below.

► **Theorem 1.3** (Symmetric sparsity bound). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be an s -sparse, symmetric polynomial with constant individual degree d , over any field \mathbb{F} . Then, every factor of f has its sparsity bounded by $\text{poly}(s)$.*

► **Remark.**

1. The exact factor-sparsity bound that we prove is $s^{O(d^2 \log d)}$. From Example 4.9, we note that s^d is a lower bound on factor-sparsity and hence also a lower bound on the time-complexity of any factoring algorithm for sparse polynomials.
2. Previous best sparsity upper bound was $s^{O(d^2 \log n)}$ due to [2], which is super polynomial even for constant d .
3. We can also work with a general (possibly *non-symmetric*) f and get the same sparsity bound for any *symmetric* factor of f (if it exists). We prove this formally in Theorem 4.6. This yields a surprising *incompressibility result*: a symmetric bounded-individual-degree polynomial g , that is s -dense, has $s^{\Omega(1)}$ -dense multiples gh , for an arbitrary nonzero polynomial h ! (See Remark 4.5)

We now give a factoring algorithm for another class of polynomials, which we call “low min-entropy” polynomials. We say that an n -dimensional vector \mathbf{v} is of δ -min-entropy if $(n - \delta)$ of its coordinates have the same value, where we consider minimum δ across all distinct elements in \mathbf{v} . We call a set of vectors A , a δ -min-entropy set, if every vector in A has min-entropy $\leq \delta$. We can identify the support of polynomial f , with the subset $\text{supp}(f) \subseteq \{0, 1, \dots, d\}^n$, as the set of exponent vectors corresponding to each monomial in f . We call f a δ -min-entropy polynomial, if $\text{supp}(f)$ is of δ -min-entropy. For eg., $f = x_1 x_2 x_3^3 x_4^5 + 2x_1^2 x_2^4 x_3^6 x_4^6 + 3x_1^7 x_2^9 x_3^8 x_4^9$ is a 2-min-entropy polynomial (also, 3-min-entropy but not 1-min-entropy). Moreover, it does *not* have symmetric-support. See Section 2 for more details on δ -min-entropy polynomials.

► **Theorem 1.4** (Low min-entropy factoring algorithm). *Let $f \in \mathbb{Q}[x_1, \dots, x_n]$ be a δ -min-entropy polynomial with individual degree d , for some constants δ, d . Let t be the maximum bit-complexity of the coefficients of f . Then, there is a deterministic algorithm that computes the complete factorization of f in at most $\text{poly}(n, t)$ -time.*

► **Remark.** The exact complexity is $\text{poly}((nd)^{d^4 \delta} \cdot t)$ -time. For a finite field $\mathbb{F} = \mathbb{F}_{p^\ell}$, we get a deterministic $\text{poly}((nd)^{d^4 \delta} \cdot \ell \log p)$ -time reduction to univariate factoring.

The factoring algorithm in Theorem 1.4 is also a direct consequence of the following sparsity bound for factors of δ -min-entropy polynomials.

► **Theorem 1.5** (Factors of low min-entropy polynomials). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a δ -min-entropy polynomial with individual degree d over any field \mathbb{F} , such that d, δ are constants. Then, every factor of f has its sparsity bounded by $\text{poly}(n)$.*

► **Remark.**

1. The exact factor-sparsity bound that we prove is $(nd)^{O(d\delta)}$, for any field \mathbb{F} . Further, our bound beats the $s^{O(d^2 \log n)}$ bound in [2], as long as f has min-entropy δ as low as $o(d \log n)$.
2. We do *not* claim that the factors are “low” min-entropy as well. For example, $f = (x_1 \cdots x_{n/2}) \cdot (x_{n/2+1} \cdots x_n)$ has both the factors with high min-entropy $n/2$, while f is itself of 0-min-entropy.

All the factor-sparsity results mentioned above crucially rely on our structure theorem stated below. It shows that if we have a δ -min-entropy set of exponent vectors, then the integral-points in its convex hull, have min-entropy at most $O(d\delta)$. Trivially, such a thing is false for \mathbb{Z} -linear-span (resp. \mathbb{Q} -linear-span) of vertices. Yet, surprisingly, a *convex*-span preserves the low min-entropy of its vertices!

► **Theorem 1.6** (Polytope entropy Theorem). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be a δ -min-entropy set, then $CS(V) \cap \mathbb{Z}^n$ is a $(2d\delta)$ -min-entropy set.*

1.2 Related works

The problem of sparse polynomial factorization was first studied in [40], where a randomized algorithm for the same was provided. The time complexity of this algorithm was polynomial in the sparsity of factors and exponential in the number of factors. Naturally, [40] raised the question of proving better sparsity bounds for factors of sparse polynomials. Even designing factoring algorithms for sparse polynomials of individual degree one or two required non-trivial insights. An efficient deterministic algorithm for factorization of sparse multilinear polynomials ($d = 1$) was provided in [31]. This result was further generalized to sparse polynomials that factorize into multilinear factors in [36]. The model of sparse multiquadratic polynomials ($d = 2$) was solved in [37].

The work of [2] utilized the interesting connection of factor-sparsity with Newton polytopes. They proved a sparsity bound of $s^{O(\log n)}$ for factors of s -sparse polynomials with constant individual degree. For symmetric polynomials in Theorem 1.3, we improve this bound considerably. The work of [2] also gives a deterministic factorization algorithm for sparse polynomials with constant individual degree which runs in $s^{O(\log n)}$ time. With our new $\text{poly}(s)$ -sparsity bound, we use their algorithm to get a deterministic $\text{poly}(s)$ -time factorization algorithm for symmetric, s -sparse polynomials, in the bounded individual degree regime.

A related and somewhat subsumed problem in polynomial factorization is that of factor closure. Given an input f in a particular representation, what is the size of factors in the same representation? The foundational work of [16, 17, 18] showed that if f is a size- s , degree- $\text{poly}(n)$ algebraic *circuit*, then its factors also have $\text{poly}(s, n)$ -sized algebraic circuits, i.e. VP is closed under factoring. [11] proved factor closure for the classes of VF, VBP, VNP with a quasi-polynomial blowup in size, and gave analogous whitebox algorithms (see Appendix B for definitions of these classes). VNP was shown to be properly closed under factoring (with only poly blowup in size) in [7]. Recently, [33] showed that size- s ABPs have factors of size $\text{poly}(s)$, thus proving factor closure for VBP class.

For algebraic formulas, [25] showed that if f is computed by a depth- Δ , size- s algebraic formula, then its factors can be computed by depth- $(\Delta + 5)$ and size $\text{poly}(s)$ formulas, provided that individual degree of f is constant. Observe that sparse polynomials can also be seen as depth-2 algebraic formulas, thus [25] showed that factors of bounded individual degree sparse polynomials can be computed by depth-7 formulas. However, [2] showed that these factors can be computed in depth-2 itself with a quasi-polynomial blowup in size. We show that if f is also symmetric, then the factors can even be computed in depth-2 efficiently, with only a polynomial blowup in size.

Another motivation for studying polynomial factorization of special classes is that improving factoring methods often lead to improvements in blackbox PIT. For eg. [24] used factoring of special circuits to give the first subexponential PIT for constant-depth circuits.

Besides being studied extensively in classical mathematics, symmetric polynomials have been investigated in the area of algebraic complexity theory as well, mostly in the context of lower bounds. A new model of depth-2 symmetric circuits was defined in [30], which studies

the complexity of determinant for this model. The complexity of elementary symmetric polynomials in the algebraic formula model is studied in [14] while [5] study a class of symmetric polynomials called Schur polynomials, also in the formula model. A strong lower bound for elementary symmetric polynomials in the model of depth-4 algebraic circuits with bounded bottom fan-in is proved in [12]. The complexity of symmetric polynomials with respect to its expression in terms of elementary symmetric polynomials is studied in [3]. A new class of algebraic circuits called symmetric algebraic circuits is studied in [8], which shows a separation of determinant from permanent in this model. Recently in [9], under a more stringent symmetry restriction in this model, lower bounds for even the determinant polynomial are shown. In this work, symmetric polynomials are studied in the context of sparse polynomial factorization.

1.3 Proof Techniques

The sparsity connection with Newton polytopes

Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a polynomial of individual degree d such that $f = \sum c_{e_1, e_2, \dots, e_n} \cdot x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}$. We define support of f as the set of exponent vectors: $\text{supp}(f) \triangleq \{(e_1, \dots, e_n) \mid c_{e_1, \dots, e_n} \neq 0\}$. Let us denote sparsity of f as $\|f\|$, which is the same as $|\text{supp}(f)|$. We define the *Newton Polytope* of f , denoted by P_f , as the convex hull (or convex-span) of all the points in $\text{supp}(f)$.

For two polytopes A and B , their *Minkowski sum* $A + B$ is defined as the set of points $\{a + b \mid a \in A, b \in B\}$. Minkowski sum is well studied in polytope literature and comes with some nice properties. The Minkowski sum $A + B$ is itself a convex polytope. Let $V(P)$ denote the set of *vertices* (corner points) of a polytope P , then one can show a certain “incompressibility” property: $|V(A + B)| \geq \max\{|V(A)|, |V(B)|\}$.

A classical fact about Newton polytopes, first observed by [27] a century ago, states that if $f = g \cdot h$, then $P_f = P_g + P_h$, where $P_g + P_h$ is the Minkowski sum of Newton polytopes of g and h . Moreover, we know that $\|f\| \geq |V(P_f)|$, since $\text{supp}(f)$ is in the convex hull of $V(P_f)$. Therefore, we are able to connect sparsity of f with a factor g as follows:

$$\|f\| \geq |V(P_f)| \geq |V(P_g)|. \quad (1)$$

Another way to think about sparsity bound problem for $f = g \cdot h$ is to determine how many monomials of g survive on multiplication with h . Equation (1) tells us that at least $|V(P_g)|$ many monomials survive. These vertices of P_g are in a sense *extremal* monomials in g that never get canceled on multiplication (with any h).

Polytope entropy Theorem

We introduce the notion of δ -min-entropy sets and polynomials in this work. In our structure theorem (Theorem 1.6), we analyze how min-entropy behaves with respect to polytopes. Suppose we are given a δ -min-entropy set V of exponent vectors from $\{0, 1, \dots, d\}^n$. The motivating question is that if all vertices (elements of V) have low min-entropy, then how large can be the min-entropy of internal points in the polytope? In this theorem, we show that min-entropy blows up only by a factor of $O(d)$ for the internal *integral-points*, which is a small factor in the bounded individual degree regime.

In order to prove this, we first design a set of symmetric hyperplane equations in Section 3 such that for each hyperplane, the entire set V lies on one side of it. Then by convexity, one can show that every point in its convex-span also lies on the same side. Secondly,

the hyperplane equations are so designed that any *integral*-point “enclosed” within these hyperplanes must have min-entropy at most $O(d\delta)$. We present this designing of “certifying” hyperplanes as a new approach for bounding sparsity of factors. Besides the factor-sparsity implications, Theorem 1.6 could be of independent interest in its own right.

Symmetric polytopes

We say that a polytope P is symmetric if for each point $\mathbf{v} = (v_1, \dots, v_n)$ in P , every permutation of \mathbf{v} is also in P . If f is a symmetric polynomial, then its Newton polytope P_f will be a symmetric polytope. Further, we observe that if a monomial (or its exponent vector) is a vertex of a symmetric polytope, then all its distinct permutations are also vertices. In other words, the vertex set $V(P_f)$ is also symmetric. Moreover, if f is sparse, then cardinality of $V(P_f)$ is also bounded. Since the vertex set is symmetric, every vector in it must contain a coordinate of “very high” frequency, otherwise $V(P_f)$ will contain a lot of distinct permutations and its size will blow up (i.e. *symmetry* \Rightarrow *low min-entropy*). Thus, $V(P_f)$ will have somewhat low min-entropy. Now, we can use our Theorem 1.6 to deduce that all the integral points in P_f have low min-entropy. In Section 4, we exploit this with additional counting arguments to prove that total number of integral points in this symmetric polytope P_f is few. In particular, we show that $|P_f \cap \mathbb{Z}^n| \leq |V(P_f)|^{O(d^2 \log d)}$, where d is the individual degree of f . If f factorizes as $f = gh$, then we have $P_f = P_g + P_h$. This means that P_f contains a translated copy of P_g . Moreover, $\text{supp}(g) \subseteq P_g$ contains only integral points and therefore, $\|g\| \leq |P_f \cap \mathbb{Z}^n| \leq |V(P_f)|^{O(d^2 \log d)}$. This implies $\|g\| \leq \|f\|^{O(d^2 \log d)}$, showing that any factor of a sparse, symmetric polynomial is also sparse (Theorem 1.3). Sparsity bound for factors of δ -min-entropy polynomials follows similar trajectory. Once we have these nice sparsity bounds, we derive efficient deterministic factoring algorithms in Section 5 using a result of [2] (Lemma 5.1).

2 Preliminaries

Notations

We use shorthand $[n]$ for the set $\{1, 2, \dots, n\}$. We denote a vector $v = (v_1, \dots, v_n)$ in short by \mathbf{v} (as a column vector). We will use the terms vector or *point* interchangeably. We use the symbol \triangleq for definitions. We will sometimes use $\mathbb{F}[\mathbf{x}]$ as short for $\mathbb{F}[x_1, \dots, x_n]$. The finite *symmetric group* on n elements, which contains all the permutations of n elements, is denoted by S_n . For a vector \mathbf{v} and a permutation $\sigma \in S_n$, we denote σ -permutation of \mathbf{v} by $\sigma \circ \mathbf{v} \triangleq (v_{\sigma(1)}, \dots, v_{\sigma(n)})$. We call a set of points symmetric, if for each point \mathbf{v} in it, $\sigma \circ \mathbf{v}$ is also in the set, for every permutation $\sigma \in S_n$. We denote the n -fold Cartesian product of a set H by H^n . We will use $\log x$ for $\log_2 x$ and $\ln x$ for $\log_e x$.

Let $f \in \mathbb{F}[\mathbf{x}]$ be an n -variate polynomial. Individual degree of a variable x_i , denoted by $\deg_{x_i}(f)$ is defined as the maximum degree of that variable in f , while *individual degree* of a polynomial is the maximum among all the individual degrees, $\max_{i \in [n]} \deg_{x_i}(f)$. We will use $\mathbf{x}^{\mathbf{e}}$ to denote the monomial $x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}$. We define $\text{coeff}(\mathbf{x}^{\mathbf{e}})(f)$ as the coefficient of monomial $\mathbf{x}^{\mathbf{e}}$ in polynomial f . We define *support* of f as $\text{supp}(f) = \{\mathbf{e} \mid \text{coeff}(\mathbf{x}^{\mathbf{e}})(f) \neq 0\}$. Let us denote *sparsity* of f as $\|f\|$, which is the same as $|\text{supp}(f)|$.

Polytopes

For a finite set of points $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{R}^n$, their *convex combination* is defined as an \mathbb{R} -linear combination of the points: $\alpha_1 \mathbf{v}_1 + \dots + \alpha_k \mathbf{v}_k$, such that $\alpha_i \geq 0$ for each $i \in [k]$ and $\sum_{i=1}^k \alpha_i = 1$. We define *convex-span* (or convex hull) $CS(\mathbf{v}_1, \dots, \mathbf{v}_k)$ as the set of all possible

convex combinations of $\mathbf{v}_i, i \in [k]$. A set $P \subseteq \mathbb{R}^n$ is called a (bounded) *polytope* if there is a finite set of points $\mathbf{v}_1, \dots, \mathbf{v}_k$ such that $P = CS(\mathbf{v}_1, \dots, \mathbf{v}_k)$. A point $\mathbf{a} \in P$ is called a *vertex* of P if it cannot be written as $\mathbf{a} = \alpha \mathbf{u} + (1 - \alpha) \mathbf{v}$ for any $\mathbf{u}, \mathbf{v} \in P \setminus \{\mathbf{a}\}$ and $\alpha \in [0, 1]$. It is equivalent to saying that vertices are *corner* points of a polytope P which cannot be expressed as convex combination of any other set of points in P . We use $V(P)$ to denote the set of vertices of P . It is easy to verify that for a polytope $P, P = CS(V(P))$. Moreover, if $P = CS(\mathbf{v}_1, \dots, \mathbf{v}_k)$ then $V(P) \subseteq \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$.

Minkowski sum of two polytopes $A, B \in \mathbb{R}^n$ is defined as the following set of points $A + B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$. A basic fact is that Minkowski sum of two polytopes is itself a polytope. The vertices of Minkowski sum have some very useful properties. It is known that every vertex of $A + B$ can be expressed *uniquely* as a sum $\mathbf{u} + \mathbf{v}$, where \mathbf{u} is a vertex of A and \mathbf{v} is a vertex of B . Additionally, one can show that $|V(A + B)| \geq \max\{|V(A)|, |V(B)|\}$ (See [2, Prop. 3.2]). We refer the readers to [41, 28] for a detailed discussion on polytopes.

For a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$, the *Newton polytope* of f is defined as $P_f \triangleq CS(\text{supp}(f))$. In this paper, we crucially exploit its *integral-points* $P'_f := P_f \cap \mathbb{Z}^n$. We denote the vertex set $V(P_f)$ by V_f . We also note that $V_f \subseteq \text{supp}(f) \subseteq \{0, \dots, d\}^n$ (integral-points). The following two facts form the backbone for the Newton polytope approach of bounding factor-sparsity.

► **Proposition 2.1** ([27]). *Let $f, g, h \in \mathbb{F}[x_1, \dots, x_n]$ be polynomials such that $f = g \cdot h$. Then $P_f = P_g + P_h$.*

► **Proposition 2.2** ([2]). *Let $f, g, h \in \mathbb{F}[x_1, \dots, x_n]$ be polynomials such that $f = g \cdot h$. Then $\|f\| \geq |V_f| \geq \max\{|V_g|, |V_h|\}$.*

Hyperplanes and Halfspaces

A hyperplane is the generalization of a line in higher dimensions. A hyperplane H is defined as $H \triangleq \{\mathbf{y} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{y} = b\}$, for some vector $\mathbf{a} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and a number $b \in \mathbb{R}$. The hyperplane divides the space into two parts $\mathbf{a}^\top \mathbf{y} \geq b$ and $\mathbf{a}^\top \mathbf{y} \leq b$. These are called halfspaces.

Symmetric polynomials and polytopes

We call $f \in \mathbb{F}[x_1, \dots, x_n]$ a *symmetric* polynomial if $f(x_1, \dots, x_n) = f(x_{\sigma(1)}, \dots, x_{\sigma(n)})$, for any permutation $\sigma \in S_n$. It is equivalent to saying: f is symmetric if and only if $\text{coeff}(x_1^{e_1} \cdots x_n^{e_n})(f) = \text{coeff}(x_1^{e_{\sigma(1)}} \cdots x_n^{e_{\sigma(n)}})(f)$, for all $\sigma \in S_n$. We call $f \in \mathbb{F}[x_1, \dots, x_n]$ a *symmetric-support* polynomial if for each monomial $x_1^{e_1} \cdots x_n^{e_n}$, we have $\text{coeff}(x_1^{e_1} \cdots x_n^{e_n})(f) \neq 0 \Rightarrow \text{coeff}(x_1^{e_{\sigma(1)}} \cdots x_n^{e_{\sigma(n)}})(f) \neq 0$, for every $\sigma \in S_n$. Note that all symmetric polynomials are also symmetric-support polynomials.

We say that a polytope P is symmetric if for each point (v_1, \dots, v_n) in $P, (v_{\sigma(1)}, \dots, v_{\sigma(n)})$ is also in P , for every permutation $\sigma \in P$. Note that Newton polytopes of symmetric-support polynomials are in fact symmetric! (See Lemma A.1)

δ -min-entropy

We say that a vector $\mathbf{v} \in \{0, 1, \dots, d\}^n$ is of δ -min-entropy if a single value $c \in \{0, \dots, d\}$ appears in exactly $(n - \delta)$ coordinates of \mathbf{v} . We consider c with the highest frequency in \mathbf{v} and hence the minimum δ . In coding theory language, δ -min-entropy vector \mathbf{v} has *Hamming distance* δ from the constant vector (c, \dots, c) . We also extend this definition of min-entropy to sets and polynomials. We call $A \subseteq \{0, 1, \dots, d\}^n$ a δ -min-entropy set, if for every vector $\mathbf{v} \in A, \mathbf{v}$ has min-entropy $\leq \delta$. We call f a δ -min-entropy polynomial, if its support set

$\text{supp}(f)$ is of δ -min-entropy. Example 4.9 gives a polynomial f with 1-min-entropy. In contrast, consider $f = x_1 \cdots x_{n/3} + x_{n/3+1} \cdots x_n$, which is a polynomial of $(n/3)$ -min-entropy but it's not $(< n/3)$ -min-entropy.

3 Polytope entropy – Theorem 1.6

Let P_f be the Newton polytope of f and V_f be its set of vertices such that $P_f = CS(V_f)$. We prove few claims below which will help us prove Theorem 1.6. We are given V_f to be a δ -min-entropy set. Let $m \triangleq n - \delta$. We first design a hyperplane $\mathbf{u}^\top \cdot \mathbf{y} + d\delta = 0$ that will help us prove very useful properties in Lemma 3.1 and Lemma 3.2 below. Define column vector $\mathbf{u} \in \{-1, +1\}^n$ such that,

$$u_i := \begin{cases} +1 & \text{if } i \leq \delta + m/2 \\ -1 & \text{otherwise.} \end{cases} \quad (2)$$

The first $\delta + m/2$ coordinates of \mathbf{u} are +1 and the remaining $n - (\delta + m/2) = m/2$ coordinates are -1.

► **Lemma 3.1** (Hyperplane cover). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be a δ -min-entropy set and \mathbf{u} be as defined in (2). Then, every point $\mathbf{y} \in V$ satisfies each of the following symmetric inequalities:*

$$(\sigma \circ \mathbf{u})^\top \cdot \mathbf{y} + d\delta \geq 0, \quad \text{for each } \sigma \in S_n. \quad (3)$$

Proof. Let \mathbf{y} be any $\leq \delta$ -min-entropy point from V ; so assume that \mathbf{y} has some majority element i with frequency $\geq m = n - \delta$, where $i \in \{0, \dots, d\}$. Let $\sigma \in S_n$ be any permutation. Define $\ell \triangleq (\sigma \circ \mathbf{u})^\top \cdot \mathbf{y}$. It suffices to show that $\ell \geq -d\delta$. We claim that to minimize ℓ , by varying $(\sigma \circ \mathbf{u})$, we can place at most δ many d 's in the coordinates corresponding to -1 and rest of the $\geq m$ coordinates must be filled by i . Since we have a total of $(\delta + m/2)$ coordinates corresponding to +1 and remaining $(m/2)$ coordinates corresponding to -1, the minimum value of ℓ in this case is $(\delta + m/2) \cdot i - (m/2 - \delta) \cdot i - (\delta) \cdot d = (2i - d)\delta \geq -d\delta$, since $i \geq 0$. Thus, $\ell \geq -d\delta$ in this configuration.

We now show why this is an optimal configuration to minimize ℓ . Note that the majority element i will always have a non-negative contribution in ℓ , since its frequency in the positions corresponding to +1 will always be greater than or equal to its frequency in the positions corresponding to -1. This is because there are only $m/2$ positions corresponding to -1 and i has frequency $\geq m$. Therefore to minimize its contribution, majority element i should be fixed to 0. To get lowest possible value from the remaining elements in \mathbf{y} , we should set all of them to d and place them in any δ positions corresponding to -1. This will give minimum value of ℓ to be $-d\delta$. Rest of the configurations can only produce higher values for ℓ . Thus, $\ell + d\delta \geq 0$ for each $\mathbf{y} \in V$ and for every $\sigma \in S_n$. ◀

The symmetry in the above hyperplane-cover inequalities helps us to argue about \mathbf{y} , by first *sorting its coordinates*. This a powerful trick, as our following central claim demonstrates.

► **Lemma 3.2** (Integral-point in the cover). *Let \mathbf{u} be as defined in (2). Let $\mathbf{y} \in \{0, 1, \dots, d\}^n$ be a point with $0 \leq y_1 \leq y_2 \leq \dots \leq y_n \leq d$ such that $\mathbf{u}^\top \cdot \mathbf{y} + d\delta \geq 0$ holds. Then, \mathbf{y} is a $(2d\delta)$ -min-entropy point.*

9:10 Sparse Polynomial Factorization

Proof. Recall $n = \delta + m$. Let us call, expectedly, the first $(\delta + m/2)$ coordinates, the “positive zone” of \mathbf{y} and the remaining $(m/2)$ coordinates, the “negative zone” of \mathbf{y} ; this corresponds to positions of $+1$ ’s and -1 ’s in \mathbf{u} respectively. By hypothesis, the positive zone has coordinates at most as large as in the negative zone. Let $\mathbf{y} =: \mathbf{y}_+ + \mathbf{y}_-$, where we define \mathbf{y}_+ and \mathbf{y}_- as

$$(\mathbf{y}_+)_j \triangleq \begin{cases} \mathbf{y}_j & \text{if } j \leq \delta + m/2 \\ 0 & \text{otherwise.} \end{cases}$$

$$(\mathbf{y}_-)_j \triangleq \begin{cases} \mathbf{y}_j & \text{if } j > \delta + m/2 \\ 0 & \text{otherwise.} \end{cases}$$

Suppose the first coordinate in the negative zone of \mathbf{y} is i for some $i \in \{0, 1, \dots, d\}$. We then claim that \mathbf{y} has at least $n - 2d\delta$ coordinates with this same value i , proving \mathbf{y} to be a $2d\delta$ -min-entropy point. Let p_+ and p_- denote the frequency of i in positive and negative zones of \mathbf{y} respectively, for some integers $p_+, p_- \geq 0$. Note that $\mathbf{u}^\top \cdot \mathbf{y} = \|\mathbf{y}_+\|_1 - \|\mathbf{y}_-\|_1$, where $\|\cdot\|_1$ is the \mathbf{L}^1 -norm. We first upper bound $\|\mathbf{y}_+\|_1$. Since \mathbf{y} is sorted, the last p_+ coordinates in positive zone must be i and all coordinates preceding it are of value at most $i - 1$. This gives us

$$\|\mathbf{y}_+\|_1 \leq (i - 1) \cdot (\delta + m/2 - p_+) + i \cdot (p_+) = i\delta + im/2 - \delta - m/2 + p_+. \quad (4)$$

Now, we lower bound $\|\mathbf{y}_-\|_1$. Since \mathbf{y} is sorted, the first p_- coordinates in negative zone must be i and all subsequent coordinates are of value at least $(i + 1)$. This gives us

$$\|\mathbf{y}_-\|_1 \geq i \cdot (p_-) + (i + 1) \cdot (m/2 - p_-) = im/2 + m/2 - p_-. \quad (5)$$

Let $l \triangleq \mathbf{u}^\top \cdot \mathbf{y} + d\delta$. By hypothesis, $l \geq 0$. Then using (4) and (5) together, we observe that

$$\begin{aligned} 0 \leq l &= \mathbf{u}^\top \cdot \mathbf{y} + d\delta = \|\mathbf{y}_+\|_1 - \|\mathbf{y}_-\|_1 + d\delta \\ 0 \leq &i\delta + im/2 - \delta - m/2 + p_+ - (im/2 + m/2 - p_-) + d\delta \\ &= (i + d)\delta - \delta - m + p_+ + p_- \\ &\leq 2d\delta - n + p_+ + p_- \quad (\text{since } i \leq d \text{ and } n = m + \delta) \\ n - 2d\delta &\leq p_+ + p_-. \end{aligned}$$

Recall that total frequency of the value i in \mathbf{y} is $p_+ + p_- \geq n - 2d\delta$. This proves that \mathbf{y} is a $(2d\delta)$ -min-entropy point; finishing the proof. (We note that the calculations in Equation (4) and Equation (5) are technically for $i \in [d - 1]$. For the corner cases of $i = 0$ or $i = d$, the same logic will work and in fact with a better lower bound on frequency of i in \mathbf{y} .) ◀

If for a set of points, each point lies on one side of a hyperplane, then all the points in their convex-span also lie on that side. We prove this lemma below.

► **Lemma 3.3** (Halfspace is convex). *Let $V = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\} \subseteq \mathbb{R}^n$ be a set of k vectors. Suppose for some $(\mathbf{a}, b) \in \mathbb{R}^n \times \mathbb{R}$ and for each $i \in [k]$, $\mathbf{a}^\top \cdot \mathbf{v}_i + b \geq 0$. Then, for any $\mathbf{v} \in CS(V)$, $\mathbf{a}^\top \cdot \mathbf{v} + b \geq 0$.*

Proof. This follows because \mathbf{v} is a convex combination of vectors in V . Let $\mathbf{v} = \alpha_1 \cdot \mathbf{v}_1 + \dots + \alpha_k \cdot \mathbf{v}_k$, where $\sum_{i=1}^k \alpha_i = 1$ and $\alpha_i \in \mathbb{R}_{\geq 0}$ for each $i \in [k]$. Thus,

$$\begin{aligned} \mathbf{a}^\top \cdot \mathbf{v} + b &= \mathbf{a}^\top \cdot \left(\sum_{i=1}^k \alpha_i \cdot \mathbf{v}_i \right) + b \\ &= \left(\sum_{i=1}^k \alpha_i \cdot \mathbf{a}^\top \cdot \mathbf{v}_i \right) + \sum_{i=1}^k \alpha_i \cdot b = \sum_{i=1}^k \alpha_i \cdot (\mathbf{a}^\top \cdot \mathbf{v}_i + b) \geq 0. \end{aligned}$$

In the second step, we use $\sum_{i=1}^k \alpha_i = 1$; while in the last step we use the hypothesis and $\alpha_i \geq 0$ for each $i \in [k]$. ◀

We are now ready to prove our structure theorem, which is restated below.

► **Theorem 1.6** (Polytope entropy Theorem). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be a δ -min-entropy set, then $CS(V) \cap \mathbb{Z}^n$ is a $(2d\delta)$ -min-entropy set.*

Proof. In Lemma 3.1, we showed that every point $\mathbf{v} \in V$ belongs to the halfspace $\mathbf{u}^\top \cdot \mathbf{v} + d\delta \geq 0$. In fact, we proved it for every permutation $\sigma \in S_n$ of \mathbf{u} . Let \mathbf{y} be any integral-point in $CS(V)$. Thus by Lemma 3.3, $(\sigma \circ \mathbf{u})^\top \cdot \mathbf{y} + d\delta \geq 0$, for every $\sigma \in S_n$.

Let $\pi \in S_n$ be the permutation which sorts \mathbf{y} , i.e. $y_{\pi(1)} \leq y_{\pi(2)} \leq \dots \leq y_{\pi(n)}$. These are integers in $\{0, \dots, d\}$ due to convexity. Since the hyperplane cover holds for every permutation, in particular it holds for $\pi^{-1} \in S_n$ also, i.e. $(\pi^{-1} \circ \mathbf{u})^\top \cdot \mathbf{y} + d\delta \geq 0$. Thus,

$$(\pi^{-1} \circ \mathbf{u})^\top \cdot \mathbf{y} + d\delta = \mathbf{u}^\top \cdot (\pi \circ \mathbf{y}) + d\delta \geq 0.$$

Now by Lemma 3.2, as $\pi \circ \mathbf{y}$ is sorted, we deduce that $\pi \circ \mathbf{y}$ is a $(2d\delta)$ -min-entropy vector. Since min-entropy of a vector does not change on permuting it, we deduce that \mathbf{y} is also a $(2d\delta)$ -min-entropy point. ◀

► **Remark.** Note that Theorem 1.6 is fairly tight, i.e. a blow-up in min-entropy of internal integral points by a factor of d is inevitable. To observe this, consider $V = \{\sigma \circ (d, 0, \dots, 0) \mid \sigma \in S_n\}$, with min-entropy $\delta = 1$. It is easy to see that the integral vector $\mathbf{v} = \sum_{i=1}^d e_i$ is in $CS(V)$, where e_i is the standard unit vector with a single 1 in position i and rest all 0. Thus, \mathbf{v} is a vector with 1 in first d coordinates and remaining all 0, and it has min-entropy exactly d , for $d \leq n/2$. Therefore, $CS(V) \cap \mathbb{Z}^n$ is of min-entropy at least $d\delta$. However, in this work we are concerned with $d = O(1)$, so even $2d\delta$ blowup is fine.

4 Factor sparsity bounds: Proof of Theorems 1.3 & 1.5

We refer the reader to Appendix A for some basic observations that will be used for the results in this section.

4.1 Polytope bounds

Using a simple counting argument below, we bound the total number of points in a given δ -min-entropy set of integral points.

► **Lemma 4.1** (min-entropy sparsity upper bound). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be a δ -min-entropy set. Then, $|V| \leq \binom{n}{\delta} \cdot (d+1)^{\delta+1}$.*

Proof. Any $\mathbf{v} \in V$ has min-entropy $\leq \delta$ and thus has at least $n - \delta$ coordinates that are equal. To specify \mathbf{v} , one needs to specify the indices of these repeated coordinates ($\leq \binom{n}{\delta}$ possibilities) and specify a total of $\delta + 1$ values for all the coordinates of \mathbf{v} ($d + 1$ possibilities for each). Hence, we deduce that $|V| \leq \binom{n}{\delta} \cdot (d+1)^{\delta+1}$. ◀

9:12 Sparse Polynomial Factorization

We now show that a symmetric set of integral points is of somewhat low min-entropy.

► **Lemma 4.2** (symmetry \Rightarrow low min-entropy). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be any symmetric set. Then V is a δ -min-entropy set, for some $\delta < 2d \cdot \log |V|$.*

Proof. Consider the smallest possible δ , for which V can be a δ -min-entropy set. By pigeonhole principle, for any point in V , there exists a coordinate-value with frequency $\geq n/(d+1)$. Therefore,

$$\delta \leq n - \frac{n}{d+1} = n \left(1 - \frac{1}{d+1}\right). \quad (6)$$

Since δ is chosen to be the minimum possible, a non-empty V contains a vector \mathbf{v} having some integral value with maximum-frequency exactly $n - \delta$. Since V is symmetric, it must also contain all the distinct S_n -permutations of this vector \mathbf{v} , which are at least $\binom{n}{\delta}$ many. This implies that $|V| \geq \binom{n}{\delta}$ and further using Lemma A.3, we get that $|V| \geq \left(\frac{n}{\delta}\right)^\delta$. Then,

$$\begin{aligned} \log |V| &\geq \delta \cdot \log \left(\frac{n}{\delta}\right) \geq \delta \cdot \log \left(\frac{n}{n \left(1 - \frac{1}{d+1}\right)}\right) \quad [\text{Using (6)}] \\ &= \delta \cdot \log \left(1 + \frac{1}{d}\right) > \delta \cdot \left(\frac{1}{d} - \frac{1}{2d^2}\right) \quad [\text{Using Lemma A.3}] \\ &\geq \frac{\delta}{2d} \quad [\text{As } d \geq 1]. \end{aligned}$$

This proves that $\delta < 2d \cdot \log |V|$. ◀

► **Remark.** We note that if we are promised $\delta \leq n/2$, then in the proof of Lemma 4.2, we get $|V| \geq \binom{n}{\delta} \geq 2^\delta$ which implies $\delta \leq \log |V|$. This will lead to an improved bound of $|V|^{O(d \log d)}$ in Theorem 4.4. In general however, δ can be higher than $n/2$. Consider for example the set of all S_n -permutations of the string $0^{n/3} \cdot 1^{n/3} \cdot 2^{n/3}$, which has $\delta = n - n/3 = 2n/3$.

We now show that convex span of a low min-entropy set has low number of integral points.

► **Theorem 4.3** (Low min-entropy polytope count). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be a δ -min-entropy set. Then, $|CS(V) \cap \mathbb{Z}^n| \leq \binom{n}{\delta}^{2d} \cdot (d+1)^{2d\delta+1}$.*

Proof. Theorem 1.6 proves that $CS(V) \cap \mathbb{Z}^n$ is a $(2d\delta)$ -min-entropy set. Note that $CS(V) \cap \mathbb{Z}^n$ is also a subset of $\{0, \dots, d\}^n$, since V is. Then by Lemma 4.1, $|CS(V) \cap \mathbb{Z}^n| \leq \binom{n}{2d\delta} \cdot (d+1)^{2d\delta+1}$. The conclusion then follows from Lemma A.3 by observing $\binom{n}{2d\delta} \leq \left(\frac{n}{\delta}\right)^{2d}$. ◀

Using the counting arguments above, we now show that the gap between number of vertices and total number of integral points in a symmetric polytope, is low.

► **Theorem 4.4** (Symmetric polytope count). *Let $V \subseteq \{0, 1, \dots, d\}^n$ be the vertices of a symmetric polytope P . Then, $|P \cap \mathbb{Z}^n| \leq |V|^{O(d^2 \log d)}$.*

Proof. Consider the smallest possible δ , for which V can be a δ -min-entropy set. Since P is a symmetric polytope, its vertex set V is also symmetric by Lemma A.2. By Lemma 4.2, the min-entropy of V is at most $\delta \triangleq O(d \cdot \log |V|)$. Then by Theorem 4.3, $|CS(V) \cap \mathbb{Z}^n| \leq \binom{n}{\delta}^{2d} \cdot (d+1)^{2d\delta+1}$. Observe that $(d+1)^{2d\delta+1} \leq d^{O(d^2 \log |V|)} = |V|^{O(d^2 \log d)}$. In Lemma 4.2, we also proved that $|V| \geq \binom{n}{\delta}$, since V is symmetric. Thus, $\binom{n}{\delta}^{2d} \leq |V|^{2d}$ and we get the desired conclusion. ◀

► **Remark 4.5.** We also prove that a symmetric polytope formed by taking convex hull of a symmetric subset E of $\{0, 1, \dots, d\}^n$ must have *many* vertices (corner points), at least $|E|^{\Omega(1/(d^2 \log d))}$ -many, to be precise. We get this *incompressibility result* by substituting $P = CS(E)$ in Theorem 4.4 and observing that $E \subseteq P \cap \mathbb{Z}^n$.

4.2 Factor-sparsity bounds

We start by proving the sparsity bound for a symmetric factor g of some sparse polynomial f , which may itself not be symmetric. To get a $\text{poly}(s)$ sparsity bound, we only require individual degree of g to be constant, while f may have arbitrary individual degree.

► **Theorem 4.6** (Symmetric factor sparsity bound). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be an s -sparse polynomial. Let g be any symmetric factor of f , with individual degree at most d . Then, g has sparsity at most $s^{O(d^2 \log d)}$.*

Proof. Let P_g be the Newton polytope of g and V_g be its vertex set. Since g is symmetric, P_g is a symmetric polytope. Now invoke Theorem 4.4 with $P = P_g$ to note that $|P_g \cap \mathbb{Z}^n| \leq |V_g|^{O(d^2 \log d)}$. Observe that $\text{supp}(g) \subseteq P_g \cap \mathbb{Z}^n$ and $|V_g| \leq |V_f| \leq s$. Therefore, $\|g\| \leq s^{O(d^2 \log d)}$. ◀

► **Remark.** In Theorem 4.6 (resp. Theorem 4.4), we do not require the whole of g (resp. P) to be symmetric, rather we only need its vertex set V_g (resp. V) to be symmetric, which is a much weaker requirement. Similarly, as will be clear below, we don't need f to be symmetric in Theorem 1.3 but only V_f to be symmetric. In these cases, we will not require use of Lemma A.2.

We will be needing the following observation for our main proofs below.

► **Observation 4.7.** *Let $f, g, h \in \mathbb{F}[x_1, \dots, x_n]$ be polynomials such that $f = g \cdot h$. Let P_f be the Newton polytope of f . Then, $\|g\| \leq |P_f \cap \mathbb{Z}^n|$.*

Proof. By Minkowski sum property, we have that $P_f = P_g + P_h$. For every $\mathbf{u} \in \text{supp}(g)$, consider a fixed point $\mathbf{v} \in \text{supp}(h)$. Observe that $\mathbf{u} + \mathbf{v} \in P_f$, since $\mathbf{u} \in P_g$ and $\mathbf{v} \in P_h$. Moreover, since the support vectors are integral, $\mathbf{u} + \mathbf{v} \in P_f \cap \mathbb{Z}^n$. In other words, $P_f \cap \mathbb{Z}^n$ contains a translated copy of $\text{supp}(g)$. This means that $\|g\| \leq |P_f \cap \mathbb{Z}^n|$. ◀

We have sufficient tools now to prove our main theorems below.

► **Theorem 1.3** (Symmetric sparsity bound). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be an s -sparse, symmetric polynomial with constant individual degree d , over any field \mathbb{F} . Then, every factor of f has its sparsity bounded by $\text{poly}(s)$.*

Proof. Since f is symmetric, its Newton polytope P_f will also be symmetric. Let V_f be the vertex set of P_f . Then, invoke Theorem 4.4 with $P = P_f$ to deduce that $|P_f \cap \mathbb{Z}^n| \leq |V_f|^{O(d^2 \log d)}$. The conclusion then follows from Observation 4.7 and $|V_f| \leq s$. ◀

We note that Theorem 1.3 does not follow from Theorem 4.6 as a symmetric polynomial f may not have symmetric factors. For example consider symmetric $f = x^3 + x^2y^2 + xy + y^3$ which factorizes as $(x^2 + y)(x + y^2)$. Each factor considered individually, is not symmetric. We also note that factors of a low min-entropy polynomial might have *high* min-entropy. But we still get a nice sparsity bound for them below.

► **Theorem 1.5** (Factors of low min-entropy polynomials). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a δ -min-entropy polynomial with individual degree d over any field \mathbb{F} , such that d, δ are constants. Then, every factor of f has its sparsity bounded by $\text{poly}(n)$.*

Proof. Let P_f be the Newton polytope of f and V_f be its vertex set. Since f is of δ -min-entropy, so is V_f , as $V_f \subseteq \text{supp}(f)$. Then by Theorem 4.3, we deduce that $|P_f \cap \mathbb{Z}^n| \leq \binom{n}{\delta}^{2d} \cdot (d+1)^{2d\delta+1} \leq (nd)^{O(d\delta)}$. The conclusion then follows from Observation 4.7. \blacktriangleleft

► **Remark.** We note that the example in Claim 4.4 of [2] is a non-symmetric polynomial with its vertices having exactly $(n/2)$ -min-entropy such that its Newton polytope has at least $n^{\Omega(\log n)}$ -many integral points. Therefore for high-entropy polynomials, a new technique is required which goes beyond counting number of internal points in the Newton polytope.

4.3 Tightness of sparsity bounds

We give two examples below where factors of a sparse polynomial have significantly high sparsity, unless the individual degree is bounded.

► **Example 4.8** ([40]). Consider the following polynomial f with individual degree d , which factorizes as $f = gh$, where

$$f = \prod_{i=1}^n (x_i^d - 1), \quad g = \prod_{i=1}^n (1 + x_i + \dots + x_i^{d-1}), \quad h = \prod_{i=1}^n (x_i - 1).$$

Observe that $\|f\| = 2^n$, while $\|g\| = d^n$. If we let $s \triangleq \|f\|$, then $\|g\| = s^{\log d}$. Over fields of characteristic 0, this is an example which exhibits highest known blowup in sparsity. Moreover, it also shows that $s^{\log d}$ is a lower bound on factor-sparsity. Note that f is a symmetric polynomial and Theorem 1.3 shows that $\|g\| \leq s^{O(d^2 \log d)}$.

► **Example 4.9** ([2]). Over finite fields we have the following polynomial which has a higher exponential blowup in factor-sparsity. Consider the following polynomial $f \in \mathbb{F}_p[x_1, \dots, x_n]$ with individual degree p , for some prime p and let $0 < d < p$. We have $f = gh$, where

$$f = x_1^p + x_2^p + \dots + x_n^p = (x_1 + x_2 + \dots + x_n)^p, \\ g = (x_1 + x_2 + \dots + x_n)^d, \quad h = (x_1 + x_2 + \dots + x_n)^{p-d}.$$

Observe that $\|f\| = n$, while $\|g\| = \binom{n+d-1}{d} \approx n^d$. If we let $s \triangleq \|f\|$, then $\|g\| \approx s^d$. The factor-sparsity bounds in this work hold for any field \mathbb{F} , finite or otherwise. Moreover, f is also symmetric and falls under the purview of Theorem 1.3, which shows that $\|g\| \leq s^{O(d^2 \log d)}$. Hence, this example shows that in Theorem 1.3, we cannot do better than s^d .

Note that this f is also a low min-entropy polynomial, in fact with $\delta = 1$ as all the exponent vectors in $\text{supp}(f)$ have $n - 1$ coordinates having the same value 0. Thus, we can use Theorem 1.5 for this f to get a factor-sparsity bound of $(np)^{O(p)}$ which is really close to the actual sparsity n^d .

5 Factoring algorithms: Proof of Theorems 1.2 & 1.4

A nice feature of the results in [2] is that once you prove a nice factor-sparsity bound, they also show how to get a deterministic factoring algorithm for sparse polynomials which runs in time polynomial in the sparsity bound proven. We restate this as Lemma 5.1 below.

► **Lemma 5.1** (Theorem 5.8 in [2]). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be an s -sparse polynomial with individual degrees at most d . Let $\xi(n, d, s)$ be the upper bound on sparsity for every factor of f . Then given f , there is a deterministic algorithm that computes complete factorization of f in $(n \cdot \xi(n, d^2, s^d))^{O(d^2)} \cdot \text{poly}(c_{\mathbb{F}}(d^2))$ field operations.*

Here $c_{\mathbb{F}}(d)$ is the best known time complexity for factoring a univariate polynomial of degree d over the field \mathbb{F} . For $\mathbb{F} = \mathbb{Q}$, $c_{\mathbb{F}}(d) \leq \text{poly}(d, t)$ where t is the maximum bit-complexity of the coefficients of f [23]. For a finite field $\mathbb{F} = \mathbb{F}_{p^\ell}$, $c_{\mathbb{F}}(d) \leq \text{poly}(\ell \cdot p, d)$ [1, 4]. In other words, the above theorem shows a deterministic reduction from multivariate to univariate factoring over any field \mathbb{F} . In [2], they showed that $\xi(n, d, s) \leq s^{O(d^2 \log n)}$ and then used Lemma 5.1 to get a factoring algorithm of $s^{\text{poly}(d) \log n}$ time complexity, which is quasi-polynomial in the bounded individual degree setting. In this work, we deal with symmetric and constant-min-entropy input polynomials, for which we show that $\xi(n, d, s) \leq (ns)^{\text{poly}(d)}$. Thus, we can use Lemma 5.1 to get polynomial time factoring algorithms in the bounded individual degree setting. The finer time complexities are discussed in the proofs of Theorem 1.2 and Theorem 1.4 below. Also, note that $\xi(n, d, s)$ is a lower bound on time complexity for sparse factoring algorithms as we output each factor as an explicit list of monomials.

► **Theorem 1.2** (Symmetric factoring algorithm). *Let $f \in \mathbb{Q}[x_1, \dots, x_n]$ be an s -sparse, symmetric polynomial with constant individual degree d . Let t be the maximum bit-complexity of the coefficients of f . Then, there is a deterministic algorithm that computes the complete factorization of f in at most $\text{poly}(s, n, t)$ -time.*

Proof. Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be an s -sparse, symmetric polynomial with individual degree d . In Theorem 1.3, we show that $\xi(n, d, s) \leq s^{O(d^2 \log d)}$. Plugging this value in Lemma 5.1, we get a deterministic factoring algorithm for f . The time complexity $T(n, s, d)$ for this algorithm is:

$$\begin{aligned} T(n, s, d) &= (n \cdot \xi(n, d^2, s^d))^{O(d^2)} \cdot \text{poly}(c_{\mathbb{F}}(d^2)) \\ &= \left(n \cdot s^{d \cdot O(d^4 \log d^2)} \right)^{O(d^2)} \cdot \text{poly}(c_{\mathbb{F}}(d^2)) \\ &= s^{O(d^7 \log d)} \cdot n^{O(d^2)} \cdot \text{poly}(c_{\mathbb{F}}(d^2)). \end{aligned}$$

This time complexity is $\text{poly}(s, n, t)$ over the field of rationals \mathbb{Q} , when d is a constant. ◀

► **Theorem 1.4** (Low min-entropy factoring algorithm). *Let $f \in \mathbb{Q}[x_1, \dots, x_n]$ be a δ -min-entropy polynomial with individual degree d , for some constants δ, d . Let t be the maximum bit-complexity of the coefficients of f . Then, there is a deterministic algorithm that computes the complete factorization of f in at most $\text{poly}(n, t)$ -time.*

Proof. Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a δ -min-entropy polynomial with individual degree d . In Theorem 1.3, we show that $\xi(n, d, s) \leq (nd)^{O(d\delta)}$. Plugging this value in Lemma 5.1, we get a deterministic factoring algorithm for f . The time complexity $T(n, s, d)$ for this algorithm is:

$$\begin{aligned} T(n, s, d) &= (n \cdot \xi(n, d^2, s^d))^{O(d^2)} \cdot \text{poly}(c_{\mathbb{F}}(d^2)) \\ &= \left(n \cdot (nd^2)^{O(d^2\delta)} \right)^{O(d^2)} \cdot \text{poly}(c_{\mathbb{F}}(d^2)) \\ &= (nd)^{O(d^4\delta)} \cdot \text{poly}(c_{\mathbb{F}}(d^2)). \end{aligned}$$

This time complexity is $\text{poly}(n, t)$ over the field of rationals \mathbb{Q} , when d, δ are constants. ◀

6 Future directions

In this work, we show a positive evidence for resolving Question 1.1 by solving it for symmetric and constant min-entropy polynomials with bounded individual degree. As discussed, the constant individual degree restriction here is necessary for sparse factorization (Examples 4.8, 4.9). We leave it as an open question to study factors of sparse polynomials with high min-entropy. For example, can one show an $(snd)^{\text{poly}(d)}$ factor-sparsity bound for s -sparse, $(\log n)$ -min-entropy polynomial of individual degree d ?

References

- 1 Elwyn R Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46(8):1853–1859, 1967.
- 2 Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *Journal of the ACM (JACM)*, 67(2):1–28, 2020. (Preliminary version in FOCS 2018).
- 3 Markus Bläser and Gorav Jindal. On the complexity of symmetric polynomials. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 4 David G Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, pages 587–592, 1981.
- 5 Prasad Chaugule, Mrinal Kumar, Nutan Limaye, Chandra Kanta Mohapatra, Adrian She, and Srikanth Srinivasan. Schur polynomials do not have small formulas if the determinant doesn't. In *Proceedings of the 35th Computational Complexity Conference*, pages 1–27, 2020.
- 6 Benny Chor and Ronald L Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.
- 7 Chi-Ning Chou, Mrinal Kumar, and Noam Solomon. Closure results for polynomial factorization. *Theory of Computing*, 15(1):1–34, 2019.
- 8 Anuj Dawar and Gregory Wilsenach. Symmetric arithmetic circuits. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 9 Anuj Dawar and Gregory Wilsenach. Lower Bounds for Symmetric Circuits for the Determinant. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2022.52.
- 10 Richard A DeMillo and Richard J Lipton. A probabilistic remark on algebraic program testing. Technical report, Georgia Inst of Tech Atlanta School of Information and Computer science, 1977.
- 11 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*, pages 1152–1165, 2018.
- 12 Hervé Fournier, Nutan Limaye, Meena Mahajan, and Srikanth Srinivasan. The shifted partial derivative complexity of elementary symmetric polynomials. In *International Symposium on Mathematical Foundations of Computer Science*, pages 324–335. Springer, 2015.
- 13 V. Guruswami and M. Sudan. Improved decoding of reed-solomon and algebraic-geometric codes. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pages 28–37, 1998. doi:10.1109/SFCS.1998.743426.
- 14 Pavel Hrubeš and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Computational Complexity*, 20(3):559–578, 2011.
- 15 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *computational complexity*, 13(1-2):1–46, 2004.

- 16 Erich Kaltofen. Uniform closure properties of p-computable functions. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 330–337, 1986.
- 17 Erich 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*, pages 443–452, 1987.
- 18 Erich Kaltofen. Factorization of polynomials given by straight-line programs. *Adv. Comput. Res.*, 5:375–412, 1989.
- 19 Erich Kaltofen. Polynomial factorization: a success story. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, pages 3–4, 2003.
- 20 Erich Kaltofen and Barry M Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.
- 21 Adam R Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 216–223, 2001.
- 22 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.
- 23 Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.
- 24 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *FOCS'21*, 2021.
- 25 Rafael Oliveira. Factors of low individual degree polynomials. *computational complexity*, 25(2):507–561, 2016.
- 26 Øystein Ore. Über höhere kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922.
- 27 AM Ostrowski. Über die bedeutung der theorie der konvexen polyeder für die formale algebra. *Jahresberichte Deutsche Math. Verein*, 20:98–99, 1921. (English translated version republished in ACM SIGSAM Bulletin, 33(1):5, 1999).
- 28 Andrzej Schinzel. *Polynomials with special regard to reducibility*, volume 77. Cambridge University Press, 2000.
- 29 Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.
- 30 Amir Shpilka. Affine projections of symmetric polynomials. *Journal of Computer and System Sciences*, 65(4):639–659, 2002.
- 31 Amir Shpilka and Ilya Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *International Colloquium on Automata, Languages, and Programming*, pages 408–419. Springer, 2010.
- 32 Amir Shpilka and Amir Yehudayoff. *Arithmetic circuits: A survey of recent results and open questions*. Now Publishers Inc, 2010.
- 33 Amit Sinhababu and Thomas Thierauf. Factorization of polynomials given by arithmetic branching programs. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 34 Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *Journal of complexity*, 13(1):180–193, 1997.
- 35 Madhu Sudan. Algebra and computation. lecture notes, 1998.
- 36 Ilya Volkovich. Deterministically factoring sparse polynomials into multilinear factors and sums of univariate polynomials. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- 37 Ilya Volkovich. On some computations on sparse polynomials. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

- 38 Joachim von zur Gathen. Who was who in polynomial factorization: 1. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, page 2, 2006.
- 39 Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.
- 40 Joachim von zur Gathen and Erich Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, 1985.
- 41 Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.
- 42 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

A Preliminary observations for Section 4

We start by proving that a symmetric polynomial will have a symmetric Newton polytope.

► **Lemma A.1** (Symmetric polynomials \Rightarrow symmetric polytopes). *Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a symmetric-support polynomial. Then the Newton polytope P_f of f is also symmetric.*

Proof. Let $\text{supp}(f) = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$. Let $\mathbf{v} \in P_f$ be an arbitrary point. We need to show that $\sigma \circ \mathbf{v} \in P_f$, for every $\sigma \in S_n$. Since $P_f = CS(\text{supp}(f))$, we can express \mathbf{v} as:

$$\mathbf{v} = \sum_{i=1}^k \alpha_i \cdot \mathbf{v}_i.$$

Then for an arbitrary $\sigma \in S_n$, observe that:

$$\sigma \circ \mathbf{v} = \sum_{i=1}^k \alpha_i \cdot \sigma \circ \mathbf{v}_i.$$

Now, since f is a symmetric-support polynomial, $\sigma \circ \mathbf{v}_i$ is also in $\text{supp}(f)$, for each $i \in [k]$. The above equation then implies that $\sigma \circ \mathbf{v} \in CS(\text{supp}(f))$, which means that $\sigma \circ \mathbf{v} \in P_f$. ◀

Next, we observe that a point is a vertex in a symmetric polytope P if and only if all its S_n permutations are also vertices in P .

► **Lemma A.2** (Vertices of symmetric polytope). *Let P be a symmetric Newton polytope with $V(P)$ being its vertex set. Then, for $\sigma \in S_n$: $\mathbf{v} \in V(P)$ if and only if $\sigma \circ \mathbf{v} \in V(P)$.*

Proof. (\Rightarrow) Suppose $V(P) = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$. Let $\mathbf{v} \in V(P)$. Consider any permutation $\sigma \in S_n$. Since P is a symmetric polytope, we at least know that $\sigma \circ \mathbf{v} \in P$. For the sake of contradiction, suppose $\sigma \circ \mathbf{v} \notin V(P)$. Thus, it is an internal point which can be expressed as a nontrivial convex combination of vertices. There exist, for $i \in [k]$, $0 \leq \alpha_i < 1$, $\sum_{i=1}^k \alpha_i = 1$ such that:

$$\begin{aligned} \sigma \circ \mathbf{v} &= \alpha_1 \cdot \mathbf{v}_1 + \dots + \alpha_k \cdot \mathbf{v}_k \\ \sigma^{-1} \circ (\sigma \circ \mathbf{v}) &= \sigma^{-1} \circ (\alpha_1 \cdot \mathbf{v}_1 + \dots + \alpha_k \cdot \mathbf{v}_k) \\ \mathbf{v} &= \alpha_1 \cdot \sigma^{-1} \circ \mathbf{v}_1 + \dots + \alpha_k \cdot \sigma^{-1} \circ \mathbf{v}_k \end{aligned}$$

Observe that $\sigma^{-1} \in S_n$ and since P is symmetric $\sigma^{-1} \circ \mathbf{v}_i \in P$ for all $i \in [k]$. This means, \mathbf{v} is a nontrivial convex combination of other points in P , which contradicts the fact that \mathbf{v} is a vertex (Section 2). Therefore, $\sigma \circ \mathbf{v}$ must also be a vertex.

(\Leftarrow) Now we wish to prove that all permutations of a non-vertex point must also be non-vertices. Let $\mathbf{v} \notin V(P)$. Then, for any $\sigma \in S_n$, we get a nontrivial convex combination:

$$\begin{aligned}\mathbf{v} &= \alpha_1 \cdot \mathbf{v}_1 + \dots + \alpha_k \cdot \mathbf{v}_k \\ \sigma \circ \mathbf{v} &= \alpha_1 \cdot \sigma \circ \mathbf{v}_1 + \dots + \alpha_k \cdot \sigma \circ \mathbf{v}_k\end{aligned}$$

Again, since P is symmetric $\sigma \circ \mathbf{v}_i \in P$ for all $i \in [k]$. Thus, $\sigma \circ \mathbf{v}$ is a nontrivial convex combination of other points, hence it must be a non-vertex. \blacktriangleleft

The lemma below mentions few standard bounds that we make use of.

► **Lemma A.3** (Counting estimates).

1. For positive integers a, b with $a \geq b$, $(a/b)^b \leq \binom{a}{b}$.
2. For positive integers a, b, c with $a \geq bc$, $\binom{a}{bc} \leq \binom{a}{c}^b$.
3. For positive real x , $\log(1+x) > \ln(1+x) > x - \frac{x^2}{2}$.

Proof. For (1), see that $\binom{a}{b} = \frac{a(a-1)\dots(a-b+1)}{b(b-1)\dots 1} \geq \left(\frac{a}{b}\right)^b$.

For (2), we make use of $\binom{a}{c+b'} \leq \binom{a}{b'} \cdot \binom{a-b'}{c} \leq \binom{a}{b'} \cdot \binom{a}{c}$. Use this b times to get $\binom{a}{bc} \leq \left(\binom{a}{c}\right)^b$.

For (3), observe that derivative of f is positive for $x > 0$, where $f = \ln(1+x) - (x - \frac{x^2}{2})$, and for $x = 0$, $f(0) = 0$. \blacktriangleleft

B Algebraic computational models

Below, we define various algebraic models for computation of a polynomial. For details, we refer the reader to the excellent survey of [32].

An *algebraic circuit* is a directed acyclic graph where computation is done bottom-up, with input leaves at the bottom and a single output node at top. The leaves are labeled with variables or field constants while rest of the nodes are either addition or multiplication nodes. The directed edges $u \rightarrow v$ are labeled with field constants, which get multiplied to the polynomial computed at node u before feeding it to node v . The in-degree of a node is called its *fan-in* and out-degree is called *fan-out*. *Size* of the circuit is simply size of the directed graph. *Depth* of the circuit is length of the longest path from a leaf to the output node. *Degree* of the circuit is maximum degree of a polynomial computed at any node in the circuit.

A size s *depth-2 $\Sigma\Pi$ circuit* computes a sum of s -many monomials. Thus, depth-2 circuits compute the class of sparse polynomials. The much more general class of $\text{poly}(n)$ -sized and $\text{poly}(n)$ degree algebraic circuits is called VP, which is considered the algebraic analog of complexity class P. The class VNP is considered the algebraic analog of complexity class NP. It is the class of polynomials which can be expressed as an exponential sum of a projection of a VP circuit family.

An algebraic circuit where fan-out of every node is one is called an *algebraic formula*. The class of polynomial sized formulas is called VF.

An *algebraic branching program (ABP)* is defined using a layered directed graph with a unique source and sink vertex. Each edge is directed from one layer to the next and has a linear polynomial as its weight. The weight of a path is product of edge weights along the path. The polynomial computed by the ABP is then simply the sum of all weighted paths from source to sink. The *length* of an ABP is the length of the longest path from source to sink and *width* of an ABP is the maximum possible number of vertices in a layer. The *size* of ABP is the product of its length and width. VBP is the class of all polynomial sized ABPs.

On Solving Sparse Polynomial Factorization Related Problems

Pranav Bisht ✉

Computer Science Department, Boston College, Chestnut Hill, MA, USA

Ilya Volkovich ✉

Computer Science Department, Boston College, Chestnut Hill, MA, USA

Abstract

In a recent result of Bhargava, Saraf and Volkovich [FOCS'18; JACM'20], the first factor sparsity bound for constant individual degree polynomials was shown. In particular, it was shown that any factor of a polynomial with at most s terms and individual degree bounded by d can itself have at most $s^{O(d^2 \log n)}$ terms. It is conjectured, though, that the “true” sparsity bound should be polynomial (i.e. $s^{\text{poly}(d)}$). In this paper we provide supporting evidence for this conjecture by presenting polynomial-time algorithms for several problems that would be implied by a polynomial-size sparsity bound. In particular, we give efficient (deterministic) algorithms for identity testing of $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{ind-deg } d}$ circuits and testing if a sparse polynomial is an exact power. Hence, our algorithms rely on different techniques.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Sparse Polynomials, Identity Testing, Derandomization, Factor-Sparsity, Multivariate Polynomial Factorization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.10

Related Version *Full Version:* <https://ecc.weizmann.ac.il/report/2022/070/> [6]

Acknowledgements The authors would like to thank the anonymous referees for their detailed comments and suggestions on the previous version of the paper.

1 Introduction

Polynomial Factorization is one of the core problems in algebraic complexity: given a multivariate polynomial $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ over a field \mathbb{F} , output all its irreducible factors. In addition to being a natural problem, its importance is highlighted by various applications such as: list decoding [28, 16], derandomization [17], cryptography [7] and others. In the seminal works of [18, 19], efficient randomized factorization algorithms were presented. Yet, coming up with an efficient *deterministic* factorization algorithm remains a long-standing open question.

Indeed, one aspect of the computational problem is the representation of the input polynomial. One natural way to represent a polynomial is by listing all its terms and coefficients. This is known as *dense* representation. Yet, even if the individual degree of every variable is bounded by a small constant d , the total number of terms can be exponentially large, reaching $(d + 1)^n$. Nonetheless, in many applications [31, 13, 3, 15] the actual number of non-zero terms in a polynomial is much smaller - $\text{poly}(n)$. Such polynomials are referred to as *sparse* polynomials, which will be the focus of our paper.

A key question that precedes the design of efficient factorization algorithms for sparse polynomial is whether a factor of a sparse polynomial is (itself) sparse. Indeed, this question was first studied by von zur Gathen and Kaltofen in [13] that gave a randomized factorization algorithm where the runtime depends on the number of terms in the output factors. In the



© Pranav Bisht and Ilya Volkovich;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 10; pp. 10:1–10:22



Leibniz International Proceedings in Informatics

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

10:2 Sparse Polynomial Factorization Related Problems

same paper they provided an example inspired by geometric series (see below) of a family of polynomials that have factors with a super-polynomial (quasi-polynomial) number of terms. We denote by $\|f\|$ the *sparsity* of f . That is, the number of non-zero terms in f .

► **Example 1** ([13]). Let $n \geq 1$. Consider the polynomial $f(\bar{x}) = \prod_{i \in [n]} (x_i^n - 1)$ which can be written as a product of $g(\bar{x}) = \prod_{i \in [n]} (1 + x_i + \dots + x_i^{n-1})$ and $h(\bar{x}) = \prod_{i \in [n]} (x_i - 1)$. Observe that $\|f\| = \|h\| = 2^n$ while $\|g\| = n^n$, resulting in a quasi-polynomial blow-up¹.

Furthermore, for fields with finite characteristics the blow-up can be significantly larger:

► **Example 2** ([29]). For a prime p , let $f \in \mathbb{F}_p[x_1, \dots, x_n]$, and let $0 < d < p$. Consider: $f(\bar{x}) = (x_1 + x_2 + \dots + x_n)^p = x_1^p + x_2^p + \dots + x_n^p$, $g(\bar{x}) = (x_1 + x_2 + \dots + x_n)^d$. Notice that g is a factor of f , but $\|f\| = n$ and $\|g\| = \binom{n+d-1}{d} = n^{\Omega(d)}$.

Based on the above, we should first try to obtain a “sparsity-bound” on factors of sparse polynomials with constant (i.e. bounded) individual degree. More formally, for some fixed d , we require that $\deg_{x_i} \leq d$, for all variables x_i . The simplest case (when $d = 1$) corresponds to the so-called *multilinear* polynomials. In [26], it was shown that a factor of an s -sparse² multilinear polynomial is itself s -sparse. Subsequently, in [30], this result was extended to the case of *multiquadratic* polynomials (i.e. when $d = 2$). In a recent work of [4], a quasi-polynomial-size sparsity bound was given for *any* fixed d . Specifically, it was shown that a factor of an s -sparse polynomial with individual degree bounded by d is $s^{O(d^2 \log n)}$ -sparse. In addition, [4] designed a factorization algorithm whose runtime is efficient in terms of the sparsity bound. As a result they obtained a deterministic quasi-polynomial-time factorization algorithm for sparse polynomials with bounded individual degree. In the same paper it was also conjectured that the “true” sparsity bound should be polynomial rather than quasi-polynomial. More formally:

► **Conjecture 3.** *There exists a universal constant $k \in \mathbb{N}$ such that for any $s, d \in \mathbb{N}$, any factor of an s -sparse polynomial with individual degree bounded by d has at most s^{dk} terms.*

In this paper we provide supporting evidence for this conjecture by presenting deterministic polynomial-time algorithms for some problems that reduce to sparse polynomial factorization. It is to be noted that invoking the aforementioned factorization algorithm of [4] with a polynomial-size sparsity bound would imply a (deterministic) *polynomial-time* algorithm for sparse polynomial factorization and hence polynomial-time algorithms for these problems. In the absence of a polynomial-size sparsity bound, we design our algorithms using new techniques.

1.1 Our Results

We will now describe our main results. In what follows, \mathbb{F} is an *arbitrary* field. (finite or otherwise).

¹ Although g is not irreducible, this issue can be resolved using standard techniques. For example, by considering the product $f + yh = (g + y)h$ for a new variable y .

² A polynomial is s -sparse, if it contains at most s non-zero terms.

1.1.1 Identity Testing for $\Sigma^{[2]}\Pi\Sigma\Pi^{[\text{ind-deg } d]}$ Circuits

The Polynomial Identity Testing (PIT) problem asks to decide whether a given input polynomial is identically zero. The input is usually given in the form of an algebraic circuit. The PIT algorithm is called *white-box* if one can look “inside” the circuit. The algorithm is called *black-box* if the circuit is given via an oracle access, where one is only allowed to evaluate the polynomial on a chosen set of input points. PIT is one of the few natural problems which have a simple efficient randomized algorithm [9, 25, 31] but lack a deterministic one. Indeed, it has been a long standing open question to come up with an efficient deterministic algorithm for this problem. We refer the reader to the full version of the paper [6] for more details.

Our first result is an efficient (deterministic) identity testing algorithm for the class of $\Sigma^{[2]}\Pi\Sigma\Pi^{[\text{ind-deg } d]}$ circuits, where a $\Sigma^{[2]}\Pi\Sigma\Pi^{[\text{ind-deg } d]}$ circuit C of size s computes a polynomial of the form:

$$C = \prod_{i=1}^r g_i + \prod_{j=1}^m h_j$$

where each polynomial (g_i and h_j) is an s -sparse polynomial with individual degree at most d (for some fixed d). Note, though, that r and m , and hence the total degree of C , can be arbitrary (i.e. polynomially) large. In particular, the polynomial computed by C may not itself be sparse. This class generalizes the model considered in [30], where $m = 1$ and the g_i -s are irreducible polynomials. For the formal definition of our circuit model and further discussion, see Section 4.1.

Observe that the identity testing problem for this circuit class reduces to polynomial factorization of sparse polynomials with bounded individual degree. Therefore, by invoking the factorization algorithm of [4], we can get an algorithm whose runtime is efficient in terms of the sparsity bound. Plugging in the best bound of [4] results in a quasi-polynomial-time algorithm. Our next result gives a *polynomial-time* algorithm for this model. In addition, our algorithm operates in the *black-box* setting, whereas the described factorization-based algorithm is a *white-box* algorithm.

► **Theorem 1.** *There exists a deterministic algorithm that given a black-box access to a $\Sigma^{[2]}\Pi\Sigma\Pi^{[\text{ind-deg } d]}$ circuit C of size s determines if $C \equiv 0$, in time $\text{poly}((sd)^{d^3}, n)$.*

An important ingredient in our algorithm is a result that links the gcd of two polynomials, their subresultant and the resultant of their coprime parts - in the **multivariate** setting. See Section 3 for the formal definitions.

► **Theorem 2.** *Let $A, B \in \mathbb{F}[x_1, x_2, \dots, x_\ell]$ be two polynomials such that $A = f \cdot g$ and $B = h \cdot g$ and let x_i be a variable. Then*

$$S_{x_i}(d, A, B) = g \cdot \text{Res}_{x_i}(f, h) \cdot \text{lc}_{x_i}(g)^{m'+n'-1}$$

here $m = \deg_{x_i}(A)$, $n = \deg_{x_i}(B)$, $d = \deg_{x_i}(g)$, $m' = \deg_{x_i}(f) = m - d$ and $n' = \deg_{x_i}(h) = n - d$. In addition:

- $\text{Res}_{x_i}(f, h)$ is the resultant of f and h w.r.t the variable x_i .
- $\text{lc}_{x_i}(g)$ is the leading coefficient of g when written as a polynomial in x_i
- And finally, $S_{x_i}(d, A, B)$ is the d -th subresultant of A and B .

To put the result in context, consider two univariate polynomials $A, B \in \mathbb{F}[x]$. A classical result in the Theory of Resultants (see e.g. [14, 12, 8]) states that:

10:4 Sparse Polynomial Factorization Related Problems

1. $\text{Res}(A, B) \equiv 0$ if and only if $\gcd(A, B)$ is non-trivial.
2. If u and v are field elements (i.e. $u, v \in \mathbb{F}$) then $\text{Res}(uA, vB) = \text{Res}(A, B) \cdot u^{\deg B} \cdot v^{\deg A}$.
3. The j -th Subresultant $S(j, A, B) \equiv 0$ whenever $j < \deg(\gcd(A, B))$.
4. There exists a non-zero field element $\alpha \in \mathbb{F}$ such that $S(j, A, B) = \alpha \cdot \gcd(A, B)$, when $j = \deg(\gcd(A, B))$.

In the multivariate setting one can always regard multivariate polynomials as polynomials in a single variable with coefficients being rational functions in the remaining variables. Yet, in this case α is no longer a mere “field element” as it can now be an arbitrary rational function in the remaining variables! From that perspective, our result can be seen as explicitly expressing α as a polynomial (and not even a rational function) in the remaining variables. We believe that this explicit relation could be of interest in its own right.

To illustrate the aforementioned problem and provide more intuition on our result, let us write the polynomials A and B in the statement of Theorem 2 as $A = (uf) \cdot (g/u)$ and $B = (uh) \cdot (g/u)$, where u is a rational function that **does not** depend on x_i . Observe that the introduction of u does not affect the degrees of x_i . We obtain the following invariant:

$$\begin{aligned} S_{x_i}(d, A, B) &= \frac{g}{u} \cdot \text{Res}_{x_i}(uf, uh) \cdot \text{lc}_{x_i} \left(\frac{g}{u} \right)^{m'+n'-1} = \\ &= \frac{g}{u} \cdot \text{Res}_{x_i}(f, h) \cdot u^{m'+n'} \cdot \frac{\text{lc}_{x_i}(g)^{m'+n'-1}}{u^{m'+n'-1}} = g \cdot \text{Res}_{x_i}(f, h) \cdot \text{lc}_{x_i}(g)^{m'+n'-1}. \end{aligned}$$

1.1.2 Exact Powers

Our next result pertains to exact powers of polynomials. A polynomial $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ is an *exact power* if there exists (another) polynomial $g \in \mathbb{F}[x_1, x_2, \dots, x_n]$ and $e \in \mathbb{N}$ such that $f = g^e$. We note that despite the rich structure, the best known sparsity bound for exact roots (i.e. $\|g\|$ in terms of $\|f\|$) is the general sparsity bound of size $s^{O(d^2 \log n)}$ by [4]. Hence, one can use the factorization algorithm of [4] to test if a given sparse polynomial is an exact power, in quasi-polynomial time. Similarly, a polynomial-size sparsity bound, even for the case of exact roots, would imply a polynomial-time algorithm for exact-power testing problem. We provide a polynomial-time algorithm for exact-power testing that **does not** rely on this sparsity bound.

► **Theorem 3.** *There is a deterministic algorithm that given a sparse polynomial $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ of individual degree d as an input, decides whether $f = g^e$ for some polynomial $g \in \mathbb{F}[x_1, x_2, \dots, x_n]$ and $e \in \mathbb{N}$, in time $\text{poly}(s^{d^2}, n)$.*

We remark that the algorithm only performs exact-power testing and **does not** output a “witness” polynomial g . Indeed, a polynomial-time algorithm that actually outputs g would imply a polynomial-size sparsity bound on exact roots! In addition, the runtime of our algorithm is polynomial in the bit-complexity of the field elements since it does not rely on univariate polynomial factorization. For instance, for finite fields we get the runtime of $\text{poly}(\log |\mathbb{F}|)$ vs $\text{poly}(|\mathbb{F}|)$.

We defer the details and proof of Theorem 3 in Section 5 of the full version of the paper [6].

1.1.3 Improved Sparsity Bounds for Co-factors of Multilinear Polynomials

Given two polynomials $f, h \in \mathbb{F}[x_1, x_2, \dots, x_n]$ such that $f = gh$, g is called a *quotient polynomial* or a *co-factor* of h . We study the problem of multilinear co-factor sparsity: suppose f is s -sparse and h is multilinear. How sparse/dense can g be? We remark that

any (even non-constructive) efficient upper bound on the sparsity of g allows us to compute g efficiently by interpolating the ratio f/h using a reconstruction algorithm for sparse polynomials (e.g. [20]) and verifying the result.

The motivation to study this problem is two-fold: first of all, by previous results (see e.g. [4]) a multilinear factor of an s -sparse polynomial (of any degree) is itself s -sparse. This suggests more structure for multilinear co-factors we could potentially exploit. Second, a polynomial-size sparsity bound on multilinear co-factors g (even when the individual degree of g is $d = 2$) would imply a polynomial-size sparsity bound for (all factors of) polynomials with individual degree $d = 3$. We note that the multicubic ($d = 3$) case is the first instance where we do not have a polynomial-size factor-sparsity bound yet. Indeed, multilinear co-factors can be seen as the “bottle-neck” for this case. The formal argument is given in Section 1.4.

To state our next result we need the following technical definition. We say that a polynomial $h \in \mathbb{F}[x_1, x_2, \dots, x_n]$ has a *unique projection of length k* if there exist k variables $x_{i_1}, x_{i_2}, \dots, x_{i_k}$ and k corresponding exponents e_1, e_2, \dots, e_k such that h has a unique monomial that contains the pattern $x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_k}^{e_k}$.

► **Theorem 4.** *Let $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a polynomial of sparsity s and individual degree at most d such that $f = gh$. Suppose, in addition, that h is a multilinear polynomial with a unique projection of length k . Then the sparsity of g is bounded by $s^{O(dk)}$.*

We remark that Example 2 with $d = p - 1$ (resulting in a lower bound of $n^{\Omega(p)}$) showcases the tightness of our result as here f is n -sparse and $h = x_1 + \dots + x_n$ has a unique projection of length 1 (e.g. x_1) which results in an upper bound of $n^{O(p)}$ for g . We can also extend Theorem 4 to the case of a co-factor of a power of a multilinear polynomial. Subsequently, we show that every multilinear s -sparse polynomial has a unique projection of length $O(\log s)$ (see Theorem 6.25 & Lemma 6.9 in the full version of the paper [6]). By plugging this result into Theorem 4, we obtain a new sparsity bound of size $s^{O(d \log s)}$ for all multilinear co-factors.

► **Corollary 4.** *Let $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a polynomial of sparsity s and individual degree at most d such that $f = gh$. Suppose, in addition, that h is a multilinear polynomial. Then the sparsity of g is bounded by $s^{O(d \log s)}$.*

The obtained bound is slightly better than the general sparsity bound of size $s^{O(d^2 \log n)}$ by [4] when $s = \text{poly}(n)$. Although our overall improvement may seem incremental (e.g. it does not allow us to “get rid” of the $\log n$ in the exponent) our main contribution here is conceptual: identifying a combinatorial property – the length of the shortest unique projection – that governs the bound on the sparsity of multilinear co-factors.

1.2 Related Work

For the sparse polynomial factorization problem, [4] have shown that factors of an s -sparse polynomial of individual degree d , have their sparsity bounded by $s^{O(d^2 \log n)}$. Currently, this is the best known bound for factor-sparsity when $d \geq 3$. For restricted classes of symmetric polynomials, Bisht and Saxena [5] recently improved this bound to $s^{O(d^2 \log d)}$.

In [13], another problem was posed alongside the sparse factorization problem, in the hope that it might be easier. This problem is referred to as *testing sparse factorization*. Given $m + 1$ sparse polynomials f, g_1, \dots, g_m , it asks to test whether $f = g_1 \dots g_m$. The work of [23] gives a polynomial-time algorithm for this problem, in the special case where every g_i is a sum of univariate polynomials. [30] gives a polynomial-time algorithm when f (and therefore every g_i) has constant individual degree and each g_i is an irreducible polynomial.

Our PIT result is connected to this problem. In Theorem 1, we give a polynomial-time algorithm to test whether $\prod_{i=1}^r f_i = \prod_{j=1}^m g_j$, where each f_i, g_j is a sparse polynomial with constant individual degree. Note that now LHS is also a product of polynomials. Moreover, there is no restriction placed on g_j -s except that they have bounded individual degree.

The depth-4 $\Sigma\Pi\Sigma\Pi$ circuit class is extremely important in the context of the PIT problem, as it is known that a polynomial-time black-box PIT for this class implies a quasi-polynomial-time black-box PIT for general VP circuits [2, 1]. For a long time, no PIT algorithm better than the trivial $d^{O(n)}$ time algorithm was known for this class, until the recent breakthrough result of Limaye et al. [21], which gives a sub-exponential time algorithm. Various restricted versions of depth-4 circuits are studied to get close to polynomial-time PIT algorithms. For example, Peleg and Shpilka [22] give a polynomial-time PIT algorithm for $\Sigma^{[3]}\Pi\Sigma\Pi^{[2]}$ circuits, where the top fan-in is 3 and the bottom fan-in is 2. Recently, Dutta et al. [10] gave a quasi-polynomial-time PIT for $\Sigma^{[k]}\Pi\Sigma\Pi^{[d]}$ circuits, where the top fan-in k and bottom fan-in d are allowed to be any fixed constants. In this model, the restriction on bottom fan-in implies that the bottom $\Sigma\Pi$ computes polynomials of total degree at most d . We give polynomial-time PIT algorithm for $\Sigma^{[2]}\Pi\Sigma\Pi^{[\text{ind-deg } d]}$ model, where the top fan-in is 2 and the bottom $\Sigma\Pi$ computes polynomials with *individual* degree at most d . We note that the individual degree restriction is much weaker than the total degree restriction. Indeed, even for the case of individual degree bounded by 1 (i.e. multilinear polynomials) the total degree can still be $\Omega(n)!$ [24] gave a polynomial-time PIT algorithm for the class of multilinear $\Sigma^{[k]}\Pi\Sigma\Pi$ circuits, with constant top fan-in k , where every gate in the circuit computes a multilinear polynomial. Yet, even a white-box polynomial-time PIT for *general* $\Sigma^{[2]}\Pi\Sigma\Pi$ circuits is still open.

Another related problem is that of *divisibility testing*, which gives two multivariate polynomials f and h and asks to decide whether h divides f . [11] gives a quasi-polynomial-time algorithm when f is sparse and h is a quadratic polynomial (and hence also sparse). We note that the quadratic restriction on h is much stronger than a constant individual degree restriction, although there is no constant degree restriction for f here. [30] gives a polynomial-time algorithm when both f, h are sparse and have constant individual degree. In the proof of Corollary 4, we solve a “search” version of the divisibility testing problem, i.e. we actually compute f/h in quasi-polynomial time, when f is sparse with constant individual degree and h is a multilinear factor of f .

1.3 Our Techniques & Proof Ideas

Let $C = \prod_{i=1}^r g_i + \prod_{j=1}^m h_j$ where g_i -s and h_j -s are s -sparse polynomials in $\mathbb{F}[x_1, x_2, \dots, x_n]$ of individual degree at most d . Clearly, if $C \equiv 0$ then it will evaluate to zero on any input. Now suppose $C \not\equiv 0$. Our goal is to find a point $\mathbf{a} \in \mathbb{F}^n$ such that $C(\mathbf{a}) \neq 0$. Our approach relies on the uniqueness of factorization property of the ring of multivariate polynomials. Specifically, we have that $\prod_{i=1}^r g_i \neq -\prod_{j=1}^m h_j$. Consequently, wlog there exists an irreducible polynomial (factor) u and $\ell > 0$ such that u^ℓ divides the LHS but does not divide the RHS. Our goal is to preserve this “situation” while reducing the number of variables. Clearly, a random projection will be sufficient. However, we wish to obtain a deterministic algorithm. To this end, we are looking for a projection that does not introduce new dependencies between factors. That is, for every i, j : if $v \mid g_i$ and $u \mid h_j$ satisfying $\gcd(u, v) = 1$ we need to ensure that $\gcd(u', v') = 1$, when u' and v' are the projections of u and v , respectively. The main tool for that is the *Resultant*. Indeed, one of the fundamental properties of the resultant is $\text{Res}(A, B) \neq 0$ if and only if $\gcd(A, B) = 1$. In the multivariate setting, this condition roughly translates into: $[\forall x_k : \text{Res}_{x_k}(u, v) \neq 0] \implies \gcd(u', v') = 1$. In other words, we

need to hit all the resultants of the form $\text{Res}_{x_k}(u, v)$ when $v \mid g_i$ and $u \mid h_j$. By definition, $\text{Res}_{x_k}(u, v)$ is a determinant of $2d \times 2d$ matrix where each entry is a coefficient of u or v . Hence, $\text{Res}_{x_k}(u, v)$ is $t^{O(d)}$ -sparse polynomial with individual degree at most $O(d^2)$, where t is an upper bound on the sparsities of u and v . Consequently, we can use a hitting set generator for sparse polynomials (e.g. [20]) to hit the resultant. As u and v are factors of s -sparse polynomials of individual degree d , the best upper by [4] will be $t = s^{O(d^2 \log s)}$. This will result in a quasi-polynomial-time algorithm.

Another idea would be to use the multiplicative properties of the resultant and hit $\text{Res}_{x_k}(h_j, g_i)$ instead. Indeed, $\text{Res}_{x_k}(h_j, g_i) \neq 0 \implies \text{Res}_{x_k}(u, v) \neq 0$ and since g_i and h_j are s -sparse, $\text{Res}_{x_k}(h_j, g_i)$ is $s^{O(d)}$ -sparse and this would get a polynomial-time algorithm. The main issue is that we could have $\text{Res}_{x_k}(u, v) \neq 0$ while $\text{Res}_{x_k}(h_j, g_i) \equiv 0$. For example, if $h_j = uf$ and $g_i = vf$ for the same polynomial f . Going back, one may ask whether we could show a better sparsity bound on $\text{Res}_{x_k}(u, v)$. While we do not quite do that, we instead show that $\text{Res}_{x_k}(u, v)$ is a factor of some $s^{O(d)}$ -sparse polynomial of individual degree at most $O(d^2)$.

As the ring of polynomials forms an integral domain, this allows us to use a polynomial-size hitting set generator for sparse polynomials.

To achieve the above goal, suppose for simplicity that $g_i = u^{a_1} \cdot v^{b_1}$ and $h_j = u^{a_2} \cdot v^{b_2}$, for some non-negative integers a_1, b_1, a_2, b_2 . If all these numbers are strictly positive, we run into the same issue we have encountered earlier. That is, $\text{Res}_{x_k}(u, v) \neq 0$ while $\text{Res}_{x_k}(h_j, g_i) \equiv 0$. To address that, we apply Theorem 2 (our key technical contribution) which allows us to “extract” the gcd. For example, if $g_i = uv^2$ and $h_j = u^2v$, we can write $g_i = v \cdot uv$ and $h_j = u \cdot uv$ and obtain that $\text{Res}_{x_k}(u, v)$ is a factor of $S_{x_k}(\deg_{x_k}(uv), g_i, h_j)$, which is an $s^{O(d)}$ -sparse polynomial (see Observation 15). However, a sole gcd extraction may be insufficient. Consider the case when $g_i = uv^2$ and $h_j = uv$. Repeating the same argument will just yield a trivial statement that $\text{Res}_{x_k}(v, 1) = 1$ is a factor of a sparse polynomial. To overcome this difficulty, we apply the previous argument on powers of g_i and h_j . That is, on $g_i^z = u^{za_1} \cdot v^{zb_1}$ and $h_j^t = u^{ta_2} \cdot v^{tb_2}$. The idea now would be to isolate the powers of u from the powers of v . Within the same example, consider $g_i^2 = u^2v^4 = v \cdot u^2v^3$ and $h_j^3 = u^3v^3 = u \cdot u^2v^3$. Now, by Theorem 2, $\text{Res}_{x_k}(u, v)$ is a factor of $S_{x_k}(\deg_{x_k}(u^2v^3), g_i^2, h_j^3)$. More generally, we show how to find appropriate “small” z and t using linear algebra.

Unfortunately, though, this could be made possible only when h_j and g_i satisfy certain “non-degeneracy” condition w.r.t u and v . More formally, when the matrix $E \triangleq \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}$ has full rank (see Lemma 20).

Our final crucial observation is that we can actually ignore “degenerate” pairs u, v . To this end, we prove a technical lemma (Lemma 7) which could be of independent interest.

1.3.1 Exact Power Testing

Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be an s -sparse polynomial of constant individual degree d . We show how to test whether $f = g^e$, for some other polynomial $g \in \mathbb{F}[x_1, \dots, x_n]$ and some $e \in \mathbb{N}$. We utilize the notion of *reverse-monic* polynomials for this result. We call a polynomial h reverse-monic, if there exists some $i \in [n]$, such that $h|_{x_i=0} = 1$. If our input polynomial f is reverse-monic, we show that g is $s^{O(d)}$ -sparse. Moreover, we also get an algorithm to compute this exact root g . We prove this in Lemmas 5.4 & 5.9 of the full version of the paper [6], using a formal expansion that can be thought of as a generalization of the Binomial Expansion: $(1+x)^{\frac{1}{e}} = \sum_{i=0}^{\infty} \binom{\frac{1}{e}}{i} x^i$. In general though, our input polynomial f may not be reverse-monic. We first convert f into a reverse-monic polynomial \hat{f} with respect to some

variable x_i , using a known standard transformation. This step only incurs a slight sparsity blow-up of s^d . One important property of this transformation is that it preserves the “exact power” structure. That is, if $f = g^e$, then $\hat{f} = h^e$, for some polynomial h . We then compute this e -th root of the reverse-monic \hat{f} , as mentioned previously.

However, we are still not quite done. It can happen that a polynomial f which was not an exact power, may become an exact power after the reverse-monic transformation. We need an additional condition to get the converse implication. We show that if both \hat{f} and $f|_{x_i=0}$ are exact powers, then we can correctly conclude that f is also an exact power. This gives us a recursive algorithm, as $f|_{x_i=0}$ is a polynomial in $(n - 1)$ variables. This procedure is described formally in Algorithm 3 of the full version of the paper [6].

1.3.2 Co-Factor Sparsity Bound

For the co-factor bounds, our results build on the division elimination techniques of [27]. Let us outline our approach. To this end, let $f, h \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be s -sparse polynomials such that $h(0, \dots, 0) = 1$ and suppose that $f = gh$ for some polynomial $g \in \mathbb{F}[x_1, x_2, \dots, x_n]$ with individual degree at most d . Consider the following formal expansion: $\frac{1}{(1-x)} = \sum_{j=0}^{\infty} x^j$.

Then we have: $g = \frac{f}{h} = \frac{f}{(1-(1-h))} = \sum_{j=0}^{\infty} f(1-h)^j$ when the equality is an equality of formal sums of monomials. The key observation is that $(1-h)$ does not contain any constants, hence total degree of every monomial in $(1-h)^j$ (and hence in every summand $f(1-h)^j$) is at least j . Consequently, we can “discard” the tail $\sum_{j=dn+1}^{\infty} f(1-h)^j$ since every monomial in g has a total degree of at most dn . Indeed, g will be formed by a subset of monomials of $\sum_{j=0}^{dn} f(1-h)^j$. This allows us to obtain an upper bound on the sparsity of g : $\|g\| \leq \sum_{j=0}^{dn} s^{j+1} \leq s^{dn+2}$. Clearly, the outlined approach has two major flaws:

1. It requires that $h(0, \dots, 0) = 1$ (or more generally, $h(0, \dots, 0) \neq 0$). And even then:
2. The obtained bound is exponential in n .

One way to address the former is by a random shift to the variable. However, this may significantly increase **both** the sparsity and the individual degree! We take a different approach. Our main observation is that the argument still works if we treat the polynomials as polynomials in “fewer” variables. Formally, let $I \subseteq [n]$ of size $|I| = k$. We can regard the polynomials as polynomials in the variables x_I with coefficient in the remaining variables. In particular, suppose that $h|_{x_I=0_I} = 1$. In this case we say that h is I -reverse monic. Observe that every monomial in $(1-h)^j$ contains at least one variable from x_I . That is, the total x_I -degree of $(1-h)^j$ is at least j and hence (as before) we can discard the tail. Yet now, g depends “only” on k variables and thus its “total” degree is kd (and not nd). This way we obtain a better upper bound on the sparsity of g , if k is “small”: $\|g\| \leq \sum_{j=0}^{kd} s^{j+1} \leq s^{kd+2}$. Of course, our approach still relies on the assumption that h is I -reverse monic for a “small” subset I . Although we are unable to lift this assumption, we can weaken it. As was noted earlier, if $h(0, \dots, 0) = \alpha \neq 0$ (i.e. when $I = [n]$) we can just divide by α as it is a field element. However, this is no longer possible for an arbitrary I (especially, if I is a small set). Yet, we observe that if $h|_{x_I=0_I} = \alpha$ and α is a non-zero *single monomial* (in the remaining variables) we can transform h into an I -reverse monic polynomial \hat{h} with the exact same sparsity. The idea is to apply the transformation $x_i = \alpha \cdot x_i$ for all $i \in I$. Note that since α is a single monomial, this transformation is reversible. Indeed, there is an 1-1 correspondence between the monomials of h and \hat{h} . Given this connection, we refer to such h as I -reverse *pseudo*-monic.

Our final ingredient is (yet) another observation that for multilinear polynomials we can weaken the assumption that h is I -reverse *pseudo*-monic further by considering *unique projections*. That is, monomials that have a “unique pattern”. Formally, we want h to have

exactly one monomial that contains the submonomial: $x_{i_1}^{e_1} x_{i_2}^{e_2} \cdots x_{i_k}^{e_k}$. We show that by “flipping” the variables in h we can transform it into another multilinear polynomial \tilde{h} which is $\{i_1, i_2, \dots, i_k\}$ -reverse pseudo-monic. As a result, $\|g\| \leq s^{kd+2}$.

This is our main conceptual contribution: the upper bound on the sparsity of a multilinear co-factor g is governed by a *combinatorial property* of the set of monomials of h : the length of the shortest unique projection. As an application, we show that every s -sparse polynomial has a unique projection of length at most $\log s + 1$, thus we obtain a new, slightly stronger, sparsity bound on co-factors of multilinear polynomials.

1.4 Multilinear co-Factor Motivation

Theorem 4 and Corollary 4 in this paper apply to the factorization scenario of $f = gh$ where f is s -sparse and h is multilinear. First of all, note that by previous results (see [4] and references within) h itself is s -sparse. So we are looking to bound the sparsity of g . As it turns out, this pattern is the “bottleneck” case for multicubic polynomials. In other words, showing a polynomial-size sparsity bound on g in this scenario would imply a polynomial-size sparsity bound on factors of general multicubic polynomials! In fact, it is sufficient to consider the case when the degree of g in every variable is exactly 2! We remark that getting polynomial-size sparsity bound is open for $d \geq 3$. The following lemma summarizes this formally.

► **Lemma 5.** *Suppose there exists an absolute constant $a \geq 1$ such that for any multicubic polynomial f : if $g \mid f$ and f/g is multilinear then $\|g\| \leq \|f\|^a$. Then for any multicubic polynomial f if $g \mid f$ then $\|g\| \leq \|f\|^a$.*

We defer the proof to Section A.

2 Preliminaries

Notations

We use the shorthand $[n]$ for the set $\{1, 2, \dots, n\}$. We denote a vector $v = (v_1, \dots, v_n)$ in short by \mathbf{v} (as a column vector). We denote the n -fold Cartesian product of a set H by H^n . The set of non-negative real numbers is denoted by $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{\geq 0}^n$ denotes the space of n -dimensional non-negative real vectors.

Let $f \in \mathbb{F}[\mathbf{x}] = \mathbb{F}[x_1, x_2, \dots, x_n]$ be an n -variate polynomial. The *individual degree* of a variable x_i in f , denoted by $\deg_{x_i}(f)$, is defined as the maximum degree of that variable in f , while the *individual degree* of f is the maximum among all the individual degrees, $\max_{i \in [n]} \deg_{x_i}(f)$. We define the *sparsity* of f as the number of non-zero terms in f . Let us denote *sparsity* of f as $\|f\|$. We say that f *depends* on a variable x_i if there exist $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ which differ only in the i -th coordinate such that $f(\mathbf{a}) \neq f(\mathbf{b})$. We define the set $\text{var}(f) \triangleq \{i \mid f \text{ depends on } x_i\}$. For $i \in [n]$, we denote by $\text{lc}_{x_i}(f)$ the *leading coefficient* of f when written as a polynomial in x_i . Formally, let $f = \sum_{j=0}^d f_j \cdot x_i^j$ such that $\forall j, f_j$ is a polynomial in rest of the variables and $f_d \neq 0$. Then $\text{lc}_{x_i}(f) \triangleq f_d$. Observe that if $f = g \cdot h$ then $\forall i \in [n] : \text{lc}_{x_i}(f) = \text{lc}_{x_i}(g) \cdot \text{lc}_{x_i}(h)$.

For a set $I \subseteq [n]$, we use x_I to denote the set of variables $\{x_i \mid i \in I\}$ and $x_{[n] \setminus I}$ to denote the set of remaining variables. We use the symbol $f|_{x_I=0_I}$ to denote the polynomial resulting from substituting 0 at all the x_I variables in f . Let $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{F}^n$. For $i \in [n]$ we define a partial assignment $\mathbf{a}_{-i} \triangleq (a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ and $f(x_i, \mathbf{a}_{-i}) \triangleq f(a_1, \dots, a_{i-1}, x_i, a_{i+1}, \dots, a_n)$.

In our setting, \mathcal{R} will be a polynomial ring, say $\mathbb{F}[x_1, \dots, x_n]$ and $\text{Res}_y(A, B)$ will be a polynomial free of y -variable. The following is the most fundamental property of the Resultant:

► **Lemma 9** (See e.g. [14, 12, 8]). *Let $A, B \in \mathbb{F}[y, x_1, \dots, x_n]$ be two polynomials. Then $\gcd_y(A, B) \neq 1$ if and only if $\text{Res}_y(A, B) \equiv 0$. That is, A and B have a non-trivial factor that depends on the variable y iff the Resultant of A, B w.r.t. y is the identically zero polynomial.*

We state the following important connection between the projection of the resultant and the GCD of projections.

► **Lemma 10.** *Let $f(y, \mathbf{x})$ and $g(y, \mathbf{x})$ be two polynomials in $\mathbb{F}[y, \mathbf{x}]$. Let $\mathbf{a} \in \mathbb{F}^n$. Then, $\text{Res}_y(f, g)(\mathbf{a}) \neq 0 \implies \gcd_y(f(y, \mathbf{a}), g(y, \mathbf{a})) = 1$.*

The proof can be found in Section A. We also require multiplicative property of the Resultant that essentially follows from the definition:

► **Lemma 11.** *Let $A, B, u, v \in \mathbb{F}[y, x_1, \dots, x_n]$ be polynomials. Then $\text{Res}_y(A, B) \mid \text{Res}_y(uA, vB)$.*

We now study few useful sub-matrices of the Sylvester matrix below.

► **Definition 12** (j -th principal resultant). *Let M_j be the submatrix of M formed by deleting last j rows of A terms, last j rows of B terms and the last $2j$ columns. We call M_j to be the j -th principal resultant of A and B . Note that $\text{Res}_y(A, B) = M = M_0$.*

We can now define the subresultant polynomial as follows.

► **Definition 13** (Subresultant). *Let M_{ij} be the $(d + e - 2j) \times (d + e - 2j)$ submatrix of Sylvester matrix M formed by deleting:*

- rows $e - j + 1$ to e (each having coefficients of $A(y)$),
- rows $d + e - j + 1$ to $d + e$ (each having coefficients of $B(y)$),
- columns $d + e - 2j$ to $d + e$, except for column $d + e - i - j$.

Note that the j -th principal resultant M_j is exactly M_{jj} .

For $0 \leq j \leq e$, the j -th subresultant of $A(y), B(y) \in \mathcal{R}[y]$ is the polynomial in $\mathcal{R}[y]$ of degree j defined by $S_y(j, A, B) = \det(M_{0j}) + \det(M_{1j}) \cdot y + \dots + \det(M_{jj}) \cdot y^j$.

We now prove our main technical result which links the gcd of two polynomials, their subresultant and the resultant of their coprime parts. Theorem 2 is a special case of this result which, we believe, could be interesting in its own right.

► **Theorem 14.** *Let $A(x), B(x) \in \mathcal{R}[x]$ be two polynomials over an arbitrary UFD \mathcal{R} . Suppose $A(x) = f(x) \cdot g(x)$ and $B(x) = h(x) \cdot g(x)$ with $\deg_x(A) = m$, $\deg_x(B) = n$, $\deg_x(g) = d$, $\deg_x(f) = m' = m - d$ and $\deg_x(h) = n' = n - d$. Then*

$$S_x(d, A, B) = g \cdot \text{Res}_x(f, h) \cdot \text{lc}_x(g)^{m'+n'-1}.$$

Due to its length and space limitation, we defer the proof of Theorem 14 to Section A.1 in the Appendix. We conclude this section making an important observation that any subresultant (and hence the Resultant) of two sparse polynomials of individual degree at most d is a sum of at most $d + 1$ determinants of $2d \times 2d$ matrices where each entry is a coefficient of a sparse polynomial and, hence is itself a (somewhat) sparse polynomial of a small individual degree.

► **Observation 15.** *Let $A, B \in \mathbb{F}[y, x_1, \dots, x_n]$ be two s -sparse polynomials with individual degrees at most d . Then for any j , $S_y(j, A, B)$ is an $(2ds)^{2d+1}$ -sparse polynomial with individual degrees at most $2d^2$.*

4 PIT for $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ Circuits

In this section we prove our main result Theorem 1. We refer the reader to the full version of the paper [6] for the formal definition of an algebraic circuit and PIT algorithm. For the purpose of black-box PIT algorithm, we require the notion of hitting set generators (HSG) or simply generators.

► **Definition 16 (Generator).** *Let \mathcal{C} be a class of n -variate polynomials. Consider $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_n) : \mathbb{F}^k \rightarrow \mathbb{F}^n$, an n -tuple of k -variate polynomials where for each $i \in [n]$, $\mathcal{G}_i \in \mathbb{F}[t_1, t_2, \dots, t_k]$. Let $f(x_1, \dots, x_n)$ be an n -variate polynomial. We define action of \mathcal{G} on polynomial f by $f(\mathcal{G}) = f(\mathcal{G}_1, \dots, \mathcal{G}_n) \in \mathbb{F}[t_1, \dots, t_k]$. We call \mathcal{G} a k -seeded generator for class \mathcal{C} if for every non-zero $f \in \mathcal{C}$, $f(\mathcal{G}) \neq 0$. Degree of generator \mathcal{G} is defined as $\deg(\mathcal{G}) \triangleq \max\{\deg(\mathcal{G}_i)\}_{i=1}^n$. We define $\mathcal{G}_{-i} \triangleq (\mathcal{G}_1, \dots, \mathcal{G}_{i-1}, \mathcal{G}_{i+1}, \dots, \mathcal{G}_n)$ and $f(x_i, \mathcal{G}_{-i}) \triangleq f(\mathcal{G}_1, \dots, \mathcal{G}_{i-1}, x_i, \mathcal{G}_{i+1}, \dots, \mathcal{G}_n)$.*

For a polynomial-time PIT algorithm, k is kept constant. A generator \mathcal{G} acts as a variable reduction map which converts an input polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ to $f(\mathcal{G}) \in \mathbb{F}[t_1, \dots, t_k]$ such that $f \equiv 0$ if and only if $f(\mathcal{G}) \equiv 0$. Let D be the degree of \mathcal{G} and d be the individual degree of f . Then \mathcal{G} gives us a brute-force hitting-set of size $(ndD)^k$ (Lemma A.2 in the full version of the paper [6]). In other words, we get a polynomial-time black-box PIT algorithm for f when k is constant, \mathcal{G} can be designed in polynomial time and its degree is also polynomially bounded.

4.1 The $\Sigma^{[k]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ Model

A size s , depth-4 $\Sigma\Pi\Sigma\Pi$ circuit computes a polynomial of the form $f = \sum_{i=1}^k \prod_{j=1}^{m_i} f_{ij}$, where f_{ij} are s -sparse polynomials for each $i \in [k], j \in [m_i]$.

For $k = 2$, even white-box PIT for $\Sigma^{[2]}\Pi\Sigma\Pi$ circuits is still open. A more restricted model is the class of $\Sigma^{[k]}\Pi\Sigma\Pi^{[d]}$ circuits, where the top fan-in k and the bottom fan-in d are constants. For a size- s circuit of this class, f_{ij} 's are s -sparse polynomials of constant total degree at most d . For $k = 3$ and $d > 2$, coming up with a polynomial PIT algorithm remains an open question here. We now introduce, what we call the $\Sigma^{[k]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ model. In the $\Sigma^{[k]}\Pi\Sigma\Pi^{[d]}$ model, the sparse polynomials f_{ij} 's have constant total degree $\leq d$. We relax this restriction to f_{ij} 's being constant *individual* degree $\leq d$ polynomials in $\Sigma^{[k]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ model. This is a more general model, since f_{ij} 's can now have much higher total degree, like $\Omega(n)$. In Section 4.2, we give a deterministic polynomial-time black-box PIT algorithm for this model when $k = 2$ and d is any constant. We also note that our PIT algorithm works for any field \mathbb{F} , while the works of [22, 10] do have certain field restrictions.

4.2 The PIT Algorithm

For a polynomial f and an irreducible polynomial u , let $e_u(f)$ denote the highest power of u in f . In other words, $f = u^{e_u(f)} \cdot g$, such that $u \nmid g$. If $u \nmid f$, then $e_u(f) = 0$. We define a polynomial Φ with respect to two non-zero polynomials P, Q as follows:

► **Definition 17.** *Let $P, Q \in \mathbb{F}[x_1, \dots, x_n]$ be two non-zero polynomials. Define the polynomial $\Phi_{P,Q} \in \mathbb{F}[x_1, \dots, x_n]$ as: $\Phi_{P,Q} \triangleq \prod_{\substack{u,v \mid PQ \\ i \in [n]}} \text{Res}_{x_i}(u, v) \cdot \prod_{i \in [n]} \text{lc}_{x_i}(P) \cdot \prod_{i \in [n]} \text{lc}_{x_i}(Q)$, where u, v*

are irreducible factors of P or Q such that $(e_u(P), e_v(P))$ interlaces with $(e_u(Q), e_v(Q))$. Moreover, we only consider non-zero multiplicands.

The next Lemma shows that a non-zero of Φ preserves non-similarity of polynomials.

► **Lemma 18.** *Let $P, Q \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be two polynomials such that $P \approx Q$ and let $\mathbf{a} \in \mathbb{F}^n$ such that $\Phi_{P,Q}(\mathbf{a}) \neq 0$. Then, there exists an $i \in [n]$ such that $P(x_i, \mathbf{a}_{-i}) \approx Q(x_i, \mathbf{a}_{-i})$.*

Proof. By our premise, we have $P \approx Q$. By uniqueness of factorization, without loss of generality, there exists an irreducible factor u of P , appearing with higher power in P than in Q . That is, $k \triangleq e_u(P) > \ell \triangleq e_u(Q) \geq 0$. Let $P = u^k \cdot G$ and $Q = u^\ell \cdot H$, for some polynomials G, H such that u does not divide either of them. Define the set $T \triangleq \{v \mid v \text{ is an irreducible factor of } H \text{ and } e_v(P) \geq e_v(Q)\}$. Then let $P = u^k \cdot \left(\prod_{v \in T} v^{e_v(P)}\right) \cdot G'$, $Q = u^\ell \cdot \left(\prod_{v \in T} v^{e_v(Q)}\right) \cdot H'$, where G', H' are the product of remaining polynomials from G and H respectively. Pick $i \in \text{var}(u)$, i.e. u depends on x_i . Note that $\text{lc}_{x_i}(P) \neq 0$, since u is a factor of P which depends on x_i . Since lc is multiplicative, we get that $\text{lc}_{x_i}(P(x_i, \mathbf{a}_{-i})) = \text{lc}_{x_i}(u(x_i, \mathbf{a}_{-i}))^k \cdot \text{lc}_{x_i}(G(x_i, \mathbf{a}_{-i}))$. From our premise, we also know that $\Phi_{P,Q}(\mathbf{a}) \neq 0$. Then by the definition of $\Phi_{P,Q}$, we get that $\text{lc}_{x_i}(P(x_i, \mathbf{a}_{-i})) \neq 0$, which implies that $\text{lc}_{x_i}(u(x_i, \mathbf{a}_{-i})) \neq 0$. Together with the fact that u has x_i -degree at least one, we conclude that $u(x_i, \mathbf{a}_{-i})$ also has x_i -degree at least one. Suppose for the sake of contradiction that $P(x_i, \mathbf{a}_{-i}) \sim Q(x_i, \mathbf{a}_{-i})$. Then:

$$u(x_i, \mathbf{a}_{-i})^{k-\ell} \cdot \left(\prod_{v \in T} v^{e_v(P)-e_v(Q)}(x_i, \mathbf{a}_{-i}) \right) \cdot G'(x_i, \mathbf{a}_{-i}) \sim H'(x_i, \mathbf{a}_{-i}).$$

Since $k > \ell$ and $\forall v \in T : e_v(P) \geq e_v(Q)$, LHS is a proper polynomial in the above equation. Moreover, $u(x_i, \mathbf{a}_{-i})$ divides LHS. Now since $\text{LHS} \sim H'(x_i, \mathbf{a}_{-i})$ and $u(x_i, \mathbf{a}_{-i})$ depends on x_i , we deduce that $H'(x_i, \mathbf{a}_{-i})$ also depends on x_i . By uniqueness of factorization, we also deduce that $u(x_i, \mathbf{a}_{-i})$ divides $H'(x_i, \mathbf{a}_{-i})$. Let $H' = v_1^{e_1} \cdot \dots \cdot v_m^{e_m}$ be the irreducible factorization of H' , for some $m \geq 1$ and $e_j \geq 1$ for all $j \in [m]$, where each v_j is irreducible. Here $e_j = e_{v_j}(Q)$ for each $j \in [m]$. Then $H'(x_i, \mathbf{a}_{-i}) = v_1(x_i, \mathbf{a}_{-i})^{e_1} \cdot \dots \cdot v_m(x_i, \mathbf{a}_{-i})^{e_m}$, where $v_j(x_i, \mathbf{a}_{-i})$'s may not be irreducible anymore due to substitution. Recall that u does not divide H and hence it does not divide H' either. Since u is irreducible, we get that $\text{gcd}_{x_i}(u, v_j) = 1$, for all $j \in [m]$. At the same time, recall that $u(x_i, \mathbf{a}_{-i})$ divides $H'(x_i, \mathbf{a}_{-i})$. Since $H'(x_i, \mathbf{a}_{-i})$ depends on x_i , this implies that $u(x_i, \mathbf{a}_{-i})$ shares a non-trivial factor with some $v_j(x_i, \mathbf{a}_{-i})$ which depends on x_i . Thus, there exists some $j \in [m]$ such that $\text{gcd}_{x_i}(u(x_i, \mathbf{a}_{-i}), v_j(x_i, \mathbf{a}_{-i})) \neq 1$. By definition of H' , $v_j \notin T$ and hence $e_{v_j}(P) < e_{v_j}(Q) = e_j$ for all $j \in [m]$. Recall that $e_u(P) > e_u(Q)$. Hence $(e_u(P), e_{v_j}(P))$ interlaces with $(e_u(Q), e_{v_j}(Q))$. By our premise, $\Phi_{P,Q}(\mathbf{a}) \neq 0$. Then by the definition of $\Phi_{P,Q}$, we get that $\text{Res}_{x_i}(u, v_j)(\mathbf{a}_{-i}) \neq 0$. By further applying Lemma 10, we deduce that $\text{gcd}_{x_i}(u(x_i, \mathbf{a}_{-i}), v_j(x_i, \mathbf{a}_{-i})) = 1$, which gives us a contradiction. Hence, $P(x_i, \mathbf{a}_{-i}) \approx Q(x_i, \mathbf{a}_{-i})$. ◀

The following is a technical lemma for future use. The proof is deferred to Section A.

► **Lemma 19.** *Let $u, v \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be two coprime and irreducible polynomials such that $\text{var}(u) \cap \text{var}(v)$ is non-empty. And suppose we have two polynomials $g = u^{a_1} \cdot v^{b_1}$ and $h = u^{a_2} \cdot v^{b_2}$, for some non-negative integers a_1, b_1, a_2, b_2 . Define $z \triangleq a_2 + b_2$ and $t \triangleq a_1 + b_1$. For any $i \in \text{var}(u) \cap \text{var}(v)$, let $W \triangleq \text{gcd}_{x_i}(g^z, h^t)$. Finally, let E be the following matrix:*

$$E \triangleq \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}. \text{ Then } \frac{g^z}{W} = \begin{cases} u^{\det(E)} & \text{if } \det(E) \geq 0 \\ v^{-\det(E)} & \text{otherwise.} \end{cases} \quad \frac{h^t}{W} = \begin{cases} v^{\det(E)} & \text{if } \det(E) \geq 0 \\ u^{-\det(E)} & \text{otherwise.} \end{cases}$$

We now show that under certain non-degeneracy condition, a resultant of two factors of sparse polynomials is itself a factor of a (somewhat) sparse polynomial.

10:14 Sparse Polynomial Factorization Related Problems

► **Lemma 20.** *Let $u \approx v \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be two irreducible polynomials. Suppose there exist s -sparse, individual degree- d polynomials $g, h \in \mathbb{F}[x_1, x_2, \dots, x_n]$ such that the matrix $E \triangleq \begin{bmatrix} e_u(g) & e_u(h) \\ e_v(g) & e_v(h) \end{bmatrix}$ has full rank. Then for any $i \in [n]$: $\text{Res}_{x_i}(u, v)$ is a factor of a non-zero $(sd)^{\mathcal{O}(d^3)}$ -sparse, $\mathcal{O}(d^4)$ -individual degree polynomial.*

Proof. Consider any $i \notin \text{var}(u) \cup \text{var}(v)$. Then $\text{Res}_{x_i}(u, v)$ is defined to be 1, which is trivially a factor of any sparse polynomial. Now consider any $i \in \text{var}(u) \setminus \text{var}(v)$. Then by definition, $\text{Res}_{x_i}(u, v) = v^{\deg_{x_i}(u)}$. Note that both $e_v(g)$ and $e_v(h)$ cannot be zero, as E has full rank. Therefore, v is factor of g or h , which are both s -sparse. Similarly, u is also a factor of g or h which implies $\deg_{x_i}(u) \leq d$. We deduce that $\text{Res}_{x_i}(u, v)$ is a factor of an s^d -sparse polynomial. Similarly, we get the same conclusion for any $i \in \text{var}(v) \setminus \text{var}(u)$. We are now left with $i \in \text{var}(u) \cap \text{var}(v)$, for which we shall prove below.

Let us write $g = u^{e_u(g)} \cdot v^{e_v(g)} \cdot A$ and $h = u^{e_u(h)} \cdot v^{e_v(h)} \cdot B$, for some polynomials $A, B \in \mathbb{F}[x_1, x_2, \dots, x_n]$ co-prime to both u and v . Let $g' = u^{e_u(g)} \cdot v^{e_v(g)}$ and $h' = u^{e_u(h)} \cdot v^{e_v(h)}$. Further, let $z = e_u(h) + e_v(h)$ and $t = e_u(g) + e_v(g)$. Consider polynomials $g^z = (g')^z \cdot A^z$ and $h^t = (h')^t \cdot B^t$. Since both g, h have individual degree d , we know that $e_u(g), e_v(g), e_u(h), e_v(h) \leq d$ and hence $s, t \leq 2d$. Pick any $i \in \text{var}(u) \cap \text{var}(v)$ and consider $\gcd_{x_i}(g^z, h^t)$. Define $W \triangleq \gcd_{x_i}((g')^z, (h')^t)$ and $Y \triangleq \gcd_{x_i}(A^z, B^t)$. Since g', h' are co-prime to both A and B , we deduce that $\gcd_{x_i}(g^z, h^t) = W \cdot Y$. By our premise, we have $\det(E) \neq 0$. Without loss of generality, let us assume $\det(E) > 0$. The other case follows similarly. Using Lemma 19, we get that $(g')^z/W = u^{\det(E)}$ and $(h')^t/W = v^{\det(E)}$. Therefore, we can write $g^z = W \cdot Y \cdot u^{\det(E)} \cdot \frac{A^z}{Y}$, $h^t = W \cdot Y \cdot v^{\det(E)} \cdot \frac{B^t}{Y}$. Note that A^z/Y and B^t/Y are proper polynomials by definition of Y . Let $\ell = \deg_{x_i}(\gcd(g^z, h^t))$. By Theorem 14, there exists $k \geq 0$ such that:

$$S_{x_i}(\ell, g^z, h^t) = W \cdot Y \cdot \text{Res}_{x_i} \left(u^{\det(E)} \cdot \frac{A^z}{Y}, v^{\det(E)} \cdot \frac{B^t}{Y} \right) \cdot \text{lc}_{x_i}(W \cdot Y)^k.$$

Since both g^z and h^t are s^{2d} -sparse with individual degree at most $2d^2$, by Observation 15, $S_{x_i}(\ell, g^z, h^t)$ is $(sd)^{\mathcal{O}(d^3)}$ -sparse with individual degree at most $8d^4$. Furthermore, observe that by definition: $\gcd_{x_i} \left(u^{\det(E)} \cdot \frac{A^z}{Y}, v^{\det(E)} \cdot \frac{B^t}{Y} \right) = 1 \implies \text{Res}_{x_i} \left(u^{\det(E)} \cdot \frac{A^z}{Y}, v^{\det(E)} \cdot \frac{B^t}{Y} \right) \neq 0 \implies S_{x_i}(\ell, g^z, h^t) \neq 0$. Finally, since $\det(E)$ is a positive integer, we use Lemma 11 to deduce that $\text{Res}_{x_i}(u, v) \mid \text{Res}_{x_i} \left(u^{\det(E)} \cdot \frac{A^z}{Y}, v^{\det(E)} \cdot \frac{B^t}{Y} \right)$. Hence, we conclude that $\text{Res}_{x_i}(u, v)$ is a factor of a non-zero $(sd)^{\mathcal{O}(d^3)}$ -sparse sub-resultant polynomial. ◀

Using the above, we conclude that while the multiplicands in the polynomial $\Phi_{P,Q}$ may not themselves be sparse, they are factors of (some) sparse polynomials. Consequently, $\Phi_{P,Q}$ can be hit by a hitting set generator for sparse polynomials.

► **Lemma 21.** *Let both $P, Q \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be products of s -sparse, individual degree- d polynomials and let \mathcal{G} be a generator for $(sd)^{\mathcal{O}(d^3)}$ -sparse, $\mathcal{O}(d^4)$ -individual degree polynomials. Then $\Phi_{P,Q}(\mathcal{G}) \neq 0$.*

Proof. Let $P = \prod_{j \in [r]} g_j$ and $Q = \prod_{k \in [m]} h_k$, where g_j, h_k are s -sparse polynomials of individual degree d , for all $j \in [r], k \in [m]$. By definition, $\Phi_{P,Q}$ has two types of multiplicands. We will show that \mathcal{G} hits both types.

For the first type, let u, v be any irreducible factors of P or Q such that $(e_u(P), e_v(P))$ interlaces with $(e_u(Q), e_v(Q))$. We wish to show that $\text{Res}_{x_i}(u, v) \neq 0 \implies \text{Res}_{x_i}(u, v)(\mathcal{G}) \neq 0$. Observe: $e_u(P) = \sum_{j=1}^r e_u(g_j)$, $e_u(Q) = \sum_{k=1}^m e_u(h_k)$, $e_v(P) = \sum_{j=1}^r e_v(g_j)$, $e_v(Q) = \sum_{k=1}^m e_v(h_k)$. By definition, each of these e-values is a non-negative integer. Therefore by Lemma 7, there exists $j \in [r], k \in [m]$ such that $E \triangleq \begin{bmatrix} e_u(g_j) & e_u(h_k) \\ e_v(g_j) & e_v(h_k) \end{bmatrix}$ has full rank. Then by Lemma 20, for any $i \in [n] : \text{Res}_{x_i}(u, v)$ is factor of some $(sd)^{\mathcal{O}(d^3)}$ -sparse, $\mathcal{O}(d^4)$ -individual degree polynomial. Since, \mathcal{G} is a generator for such polynomials, we deduce that $\text{Res}_{x_i}(u, v)(\mathcal{G}) \neq 0$.

For the second type, by multiplicative property of lc , we know that for any $i \in [n] : \text{lc}_{x_i}(P) = \prod_{j \in [r]} \text{lc}_{x_i}(g_j)$ and $\text{lc}_{x_i}(P)(\mathcal{G}) = \prod_{j \in [r]} \text{lc}_{x_i}(g_j)(\mathcal{G})$. Note that $\text{lc}_{x_i}(g_j)$ is also s -sparse with individual degree d . Hence, $\text{lc}_{x_i}(g_j)(\mathcal{G}) \neq 0$, for all $j \in [r], i \in [n]$. This implies $\text{lc}_{x_i}(P)(\mathcal{G}) \neq 0$, for all $i \in [n]$ (whenever $\text{lc}_{x_i}(P) \neq 0$). Similarly, we can show $\text{lc}_{x_i}(Q)(\mathcal{G}) \neq 0$, for all $i \in [n]$. We conclude that $\Phi_{P,Q}(\mathcal{G}) \neq 0$. ◀

By combining the result with Lemma 18 we obtain the following corollary.

► **Corollary 22.** *Let $P, Q \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be products of s -sparse, individual degree- d polynomials such that $P \approx Q$. Let \mathcal{G} be a generator for $(sd)^{\mathcal{O}(d^3)}$ -sparse, $\mathcal{O}(d^4)$ -individual degree polynomials. Then there exists an $i \in [n]$ such that $P(x_i, \mathcal{G}_{-i}) \approx Q(x_i, \mathcal{G}_{-i})$.*

Finally, we describe the black-box PIT algorithm for $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ circuits. The correctness and the time complexity follow from Corollary 22. Hence, Theorem 1 follows from this Lemma. Due to space limitation we give the proof of Corollary 22 and the detailed analysis of Algorithm 1 in Section A.3.

► **Lemma 23.** *There exists a deterministic algorithm that given n, d, s and a black-box access to a $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ circuit C of size s determines if $C \equiv 0$, in time $\text{poly}((sd)^{d^3}, n)$. Algorithm 1 provides the outline.*

■ **Algorithm 1** Black-box PIT algorithm for class $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$.

Input: A black-box access to a polynomial $f(x_1, \dots, x_n) \in \Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$

Output: “ZERO”, if f is identically zero and “NON-ZERO”, otherwise.

- 1 Invoke [20] to get generator \mathcal{G} of seed-length 1 for n -variate polynomials of sparsity $\leq (sd)^{\mathcal{O}(d^3)}$ and individual degree $\leq \mathcal{O}(d^4)$.
 - 2 **for** $i \leftarrow 1$ to n **do**
 - 3 Compute the bivariate polynomial $f(x_i, \mathcal{G}_{-i})$.
 - 4 Do brute-force black-box PIT for $f(x_i, \mathcal{G}_{-i})$.
 - 5 **if** $f(x_i, \mathcal{G}_{-i}) \neq 0$ **then return** “NON-ZERO”.
 - 6 **return** “ZERO”.
-

5 Future Directions

A lot of interesting open problems arise in the context of this work:

- Design a polynomial-time PIT algorithm for $\Sigma^{[k]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ circuits with bounded k and d , for $k \geq 3$. To the best of our knowledge, the smallest open case is $k = 3$ and $d = 1!$
- Prove a polynomial-size sparsity bound (Conjecture 3) even for the special cases like exact-roots, multilinear co-factors.
 - In particular, improve the sparsity bound in Corollary 4. Ideally, get rid of the $\log s$ term in the exponent. One can start by studying the structure of polynomials with non-constant or log-sized unique projections.

- Can we show that $\text{Res}_{x_i}(u, v)$ is actually sparse (or “somewhat sparse”) under the premises of Lemma 20? This claim will be implied by a polynomial-size sparsity bound.

References

- 1 M. Agrawal, S. Ghosh, and N. Saxena. Bootstrapping variables in algebraic circuits. *Proceedings of the National Academy of Sciences*, 116(17):8107–8118, 2019.
- 2 M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 67–75, 2008.
- 3 M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 301–309, 1988.
- 4 V. Bhargava, S. Saraf, and I. Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *J. ACM*, 67(2):8:1–8:28, 2020.
- 5 P. Bisht and N. Saxena. Derandomization via symmetric polytopes: Poly-time factorization of certain sparse polynomials. In *Proceedings of the 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2022. URL: <https://www.cse.iitk.ac.in/users/nitin/papers/symmetricSparse.pdf>.
- 6 P. Bisht and I. Volkovich. On solving sparse polynomial factorization related problems. *Electron. Colloquium Comput. Complex.*, TR22-070, 2022. URL: <https://eccc.weizmann.ac.il/report/2022/070>.
- 7 B. Chor and R. L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.
- 8 D. A. Cox, J. Little, and D. O’Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (4. ed.)*. Undergraduate texts in mathematics. Springer, 2015.
- 9 R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.
- 10 P. Dutta, P. Dwivedi, and N. Saxena. Deterministic identity testing paradigms for bounded top-fanin depth-4 circuits. In *36th Conference on Computational Complexity (CCC 2021)*, volume 5, page 9, 2021.
- 11 M. A. Forbes. Deterministic divisibility testing via shifted partial derivatives. In *FOCS*, 2015.
- 12 J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.
- 13 J. von zur Gathen and E. Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, 1985. doi:10.1016/0022-0000(85)90044-3.
- 14 K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for computer algebra*. Kluwer, 1992.
- 15 E. Grigorescu, K. Jung, and R. Rubinfeld. A local decision test for sparse polynomials. *Inf. Process. Lett.*, 110(20):898–901, 2010. doi:10.1016/j.ipl.2010.07.012.
- 16 V. Guruswami and M. Sudan. Improved decoding of reed-solomon codes and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.
- 17 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.
- 18 E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness in Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press Inc., Greenwich, Connecticut, 1989.
- 19 E. Kaltofen and B. M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. of Symbolic Computation*, 9(3):301–320, 1990.

- 20 A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–223, 2001.
- 21 N. Limaye, S. Srinivasan, and S. Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *FOCS 2021*, 2022.
- 22 S. Peleg and A. Shpilka. Polynomial time deterministic identity testing algorithm for $\Sigma^{[3]}\Pi\Sigma\Pi^{[2]}$ circuits via edelstein–kelly type theorem for quadratic polynomials. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 259–271, 2021.
- 23 C. Saha, R. Saptharishi, and N. Saxena. A case of depth-3 identity testing, sparse factorization and duality. *Computational Complexity*, 22(1):39–69, 2013. doi:10.1007/s00037-012-0054-4.
- 24 S. Saraf and I. Volkovich. Blackbox identity testing for depth-4 multilinear circuits. *Combinatorica*, 38(5):1205–1238, 2018.
- 25 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.
- 26 A. Shpilka and I. Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *Automata, Languages and Programming, 37th International Colloquium (ICALP)*, pages 408–419, 2010. Full version at <https://eccc.weizmann.ac.il/report/2010/036>.
- 27 V. Strassen. Vermeidung von divisionen. *J. of Reine Angew. Math.*, 264:182–202, 1973.
- 28 M. Sudan. Decoding of reed solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.
- 29 I. Volkovich. Deterministically factoring sparse polynomials into multilinear factors and sums of univariate polynomials. In *APPROX-RANDOM*, pages 943–958, 2015.
- 30 I. Volkovich. On some computations on sparse polynomials. In *APPROX-RANDOM*, pages 48:1–4:21, 2017.
- 31 R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 216–226, 1979.

A Missing Proofs

Proof of Lemma 5. We prove the following claim: Let $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a multicubic polynomial such that $f = uv$ and $v \neq 0$. Then $\|u\| \leq \|f\|^a$. Note that the claim also covers the case when $f = u \equiv 0$. The proof is by induction on n (the number of variables in f). The base case is when $n = 0$ (i.e. $u, f, v \in \mathbb{F}$) where the claim follows trivially. Suppose $n \geq 1$. We have the following cases to consider:

- There exists a variable x_i s.t. $\deg_{x_i}(u) \geq 1$ but $\deg_{x_i}(v) = 0$. Let $1 \leq d \leq 3$ be the degree of x_i in u . In this case we can write:

$$(u_d x_i^d + \dots + u_0)v = uv = f = f_d x_i^d + \dots + f_0.$$

Here, u_j, f_j and v do not depend on x_i . Formally: $f_j = u_j v$ for $j \in \{0, \dots, d\}$. By the induction hypothesis, we have that $\|u_j\| \leq \|f_j\|^a$ for $j \in \{0, \dots, d\}$ and hence:

$$\|u\| = \sum_{j=0}^d \|u_j\| \leq \sum_{j=0}^d \|f_j\|^a \leq \left(\sum_{j=0}^d \|f_j\| \right)^a = \|f\|^a.$$

- There exists a variable x_i s.t. $\deg_{x_i}(v) \geq 1$, but $\deg_{x_i}(u) = 0$. Pick $\alpha \in \mathbb{F}$ such that $v|_{x_i=\alpha} \neq 0$. We have that:

$$u \cdot v|_{x_i=\alpha} = u|_{x_i=\alpha} \cdot v|_{x_i=\alpha} = f|_{x_i=\alpha}.$$

By the induction hypothesis: $\|u\| \leq \|f|_{x_i=\alpha}\|^a \leq \|f\|^a$.

10:18 Sparse Polynomial Factorization Related Problems

- There exists a variable x_i s.t. $\deg_{x_i}(u) = 1$. Wlog $\deg_{x_i}(v) \geq 1$. We can write

$$(u_1x_i + u_0)(v_dx_i^d + \dots + v_ex_i^e) = uv = f = (f_{d+1}x_i^{d+1} + \dots + f_ex_i^e).$$

Here, $d > e$ and $v_d, v_e \neq 0$. In particular, we have that $u_1v_d = f_{d+1}$ and $u_0v_e = f_e$. By the induction hypothesis: $\|u\| = \|u_1\| + \|u_0\| \leq \|f_{d+1}\|^a + \|f_e\|^a \leq \|f\|^a$.

- WLOG we are left with the case that for each $i \in [n]$ we have that: $\deg_{x_i}(u) = 2$ and $\deg_{x_i}(v) = 1$. Based on our assumption, in this case $\|u\| \leq \|f\|^a$ and we are done. ◀

We state the following important connection between projection of resultant and resultant of projections.

► **Lemma 24.** *Let $f, g \in \mathbb{F}[y, \mathbf{x}]$ be two polynomials and let $\mathbf{a} \in \mathbb{F}^n$. Then,*

$$\text{Res}_y(f, g)(\mathbf{a}) \neq 0 \implies \text{Res}_y(f(\mathbf{a}), g(\mathbf{a})) \neq 0.$$

Proof. Let $d \triangleq \deg_y(f)$, $e \triangleq \deg_y(g)$, $r \triangleq \deg_y(f(\mathbf{a}))$ and $t \triangleq \deg_y(g(\mathbf{a}))$. Then with some easy determinant calculations, one can show that:

$$\text{Res}_y(f, g)(\mathbf{a}) = \begin{cases} \text{Res}_y(f(\mathbf{a}), g(\mathbf{a})) & r = d, t = e \\ (\text{lc}_y(f)(\mathbf{a}))^{e-t} \cdot \text{Res}_y(f(\mathbf{a}), g(\mathbf{a})) & r = d, t < e \\ (-1)^{e(d-r)} \cdot (\text{lc}_y(g)(\mathbf{a}))^{d-r} \cdot \text{Res}_y(f(\mathbf{a}), g(\mathbf{a})) & r < d, t = e \\ 0 & r < d, t < e \end{cases}$$

Note that if $\text{Res}_y(f, g)(\mathbf{a}) \neq 0$, then $\text{Res}_y(f(\mathbf{a}), g(\mathbf{a}))$ divides it and hence the conclusion follows. ◀

Lemma 10 follows from Lemma 9 and Lemma 24.

Proof of Lemma 19. We have that: $g^z = (u^{a_1} \cdot v^{b_1})^z = u^{a_1a_2+a_1b_2} \cdot v^{a_2b_1+b_1b_2}$ and $h^t = (u^{a_2} \cdot v^{b_2})^t = u^{a_1a_2+a_2b_1} \cdot v^{a_1b_2+b_1b_2}$. If $\det(E) \geq 0$, then $a_1b_2 \geq a_2b_1$ and consequently $W = u^{a_1a_2+a_2b_1} \cdot v^{a_2b_1+b_1b_2}$. In that case, $g^z/W = u^{a_1b_2-a_2b_1} = u^{\det(E)}$ and $h^t/W = v^{a_1b_2-a_2b_1} = v^{\det(E)}$. Otherwise, if $\det(E) < 0$, then $a_2b_1 > a_1b_2$ and consequently $W = u^{a_1a_2+a_1b_2} \cdot v^{a_1b_2+b_1b_2}$. Then $g^z/W = v^{a_2b_1-a_1b_2} = v^{-\det(E)}$ and $h^t/W = u^{a_2b_1-a_1b_2} = u^{-\det(E)}$. ◀

A.1 Proof of Theorem 14

In this section we give the proof of Theorem 14, from which Theorem 2 follows. We start by stating below some known results in the theory of subresultants, which will be useful for us.

► **Lemma 25** (Lem 7.1 of [14]). *Let $A(x), B(x) \in \mathcal{R}[x]$ be two polynomials over an arbitrary UFD \mathcal{R} . Let \mathcal{K} be the field of fractions of \mathcal{R} . Suppose $A(x) = Q(x) \cdot B(x) + R(x)$, for some polynomials $Q, R \in \mathcal{K}[x]$ such that $\deg_x(A) = m$, $\deg_x(B) = n$, $\deg_x(Q) = m - n$, $\deg_x(R) = k$ and $m \geq n > k$. Let b and r denote the leading coefficients of $B(x)$ and $R(x)$ respectively. Then*

$$S_x(j, A, B) = (-1)^{(m-j)(n-j)} \times \begin{cases} b^{m-k} \cdot S_x(j, B, R) & 0 \leq j < k \\ b^{m-k} \cdot r^{n-k-1} \cdot R(x) & j = k \\ 0 & k < j < n - 1 \\ b^{m-n+1} \cdot R(x) & j = n - 1. \end{cases}$$

That is, $S_x(j, A, B)$ equals to one of the above four expressions multiplied by the corresponding sign $(-1)^{(m-j)(n-j)}$.

For the sake of completeness, we restate Theorem 14 below.

► **Theorem.** Let $A(x), B(x) \in \mathcal{R}[x]$ be two polynomials over an arbitrary UFD \mathcal{R} . Suppose $A(x) = f(x) \cdot g(x)$ and $B(x) = h(x) \cdot g(x)$ with $\deg_x(A) = m$, $\deg_x(B) = n$, $\deg_x(g) = d$, $\deg_x(f) = m' = m - d$ and $\deg_x(h) = n' = n - d$. Then

$$S_x(d, A, B) = g \cdot \text{Res}_x(f, h) \cdot \text{lc}_x(g)^{m'+n'-1}.$$

Proof of Theorem 14. Let \mathcal{K} be the field of fractions of UFD \mathcal{R} . Consider Euclidean division of A by B in $\mathcal{K}[x]$ so that we get $A(x) = Q(x) \cdot B(x) + R(x)$, for some polynomials $Q, R \in \mathcal{K}[x]$ such that $\deg_x(R) < \deg_x(B)$. Note that since g divides both A and B , it must also divide R . Therefore, $R = g \cdot p$ for some polynomial $p(x) \in \mathcal{K}[x]$. Thus, we also get

$$f(x) = Q(x) \cdot h(x) + p(x) \quad (\text{A.1})$$

Let $\deg_x(R) = k$ for some $k < n$ and let $\deg_x(p) = k' = k - d$. Now, we prove the theorem by induction on $\deg_x(p)$.

Base case. $\deg_x(p) = k' = 0$. In other words, $\deg_x(R) = k = d$. Thus using second case of Lemma 25, we get that:

$$\begin{aligned} S_x(d, A, B) &= (-1)^{(m-d)(n-d)} \cdot b^{m-k} \cdot r^{n-k-1} \cdot R \\ &= (-1)^{m'.n'} \cdot \text{lc}_x(h)^{m-k} \cdot \text{lc}_x(g)^{m-k} \cdot \text{lc}_x(p)^{n-k-1} \cdot \text{lc}_x(g)^{n-k-1} \cdot pg \\ &= (-1)^{m'.n'} \cdot g \cdot \text{lc}_x(h)^{m-k} \cdot \text{lc}_x(p)^{n-k} \cdot \text{lc}_x(g)^{m+n-2k-1} \\ S_x(d, A, B) &= (-1)^{m'.n'} \cdot g \cdot \text{lc}_x(h)^{m-k} \cdot \text{lc}_x(p)^{n-k} \cdot \text{lc}_x(g)^{m'+n'-1} \end{aligned} \quad (\text{A.2})$$

The second last step above follows because $p = \text{lc}_x(p)$ when $\deg_x(p) = 0$. Now, we shall compute $\text{Res}_x(f, h)$. Note that $\text{Res}_x(f, h) = S_x(0, f, h)$ by definition of subresultant. Considering (A.1) with $\deg_x(p) = 0$, we can use second case of Lemma 25 to get:

$$\begin{aligned} S_x(0, f, h) &= (-1)^{(\deg_x(f)-0) \cdot (\deg_x(h)-0)} \cdot \text{lc}_x(h)^{\deg_x(f)-\deg_x(p)} \cdot \text{lc}_x(p)^{\deg_x(h)-\deg_x(p)-1} \cdot p \\ &= (-1)^{m'.n'} \cdot \text{lc}_x(h)^{m'} \cdot \text{lc}_x(p)^{n'-1} \cdot p \quad [\text{as } \deg_x(p) = 0] \\ &= (-1)^{m'.n'} \cdot \text{lc}_x(h)^{m'} \cdot \text{lc}_x(p)^{n'} \quad [\text{as } p = \text{lc}_x(p)] \\ \text{Res}_x(f, h) &= (-1)^{m'.n'} \cdot \text{lc}_x(h)^{m-k} \cdot \text{lc}_x(p)^{n-k} \end{aligned} \quad (\text{A.3})$$

(A.2) and (A.3) together yield $S_x(d, A, B) = g \cdot \text{Res}_x(f, h) \cdot \text{lc}_x(g)^{m'+n'-1}$ for the base case.

Induction step. Now, we assume $\deg_x(p) = k' > 1$. In other words, $\deg_x(R) = k > d$. Therefore, by first case of Lemma 25:

$$\begin{aligned} S_x(d, A, B) &= (-1)^{(m-d)(n-d)} \cdot b^{m-k} \cdot S_x(d, B, R) \\ &= (-1)^{m'.n'} \cdot \text{lc}_x(h)^{m-k} \cdot \text{lc}_x(g)^{m-k} \cdot S_x(d, B, R) \end{aligned} \quad (\text{A.4})$$

Now consider Euclidean division of B by R in $\mathcal{K}[x]$ to get

$$B(x) = Q'(x) \cdot R(x) + R'(x) \quad (\text{A.5})$$

for some polynomial $R'(x) \in \mathcal{K}[x]$ with $\deg_x(R') < \deg_x(R)$. Since g divides both B and R , we deduce that g must also divide R' . Let $R' = g \cdot p'$ for some polynomial $p' \in \mathcal{K}[x]$. Thus from (A.5), we also get

$$h(x) = Q'(x) \cdot p(x) + p'(x) \quad (\text{A.6})$$

10:20 Sparse Polynomial Factorization Related Problems

In (A.5) since $\deg_x(R') < \deg_x(R)$ or equivalently $\deg_x(p') < \deg_x(p)$, we can use induction hypothesis to deduce that,

$$S_x(d, B, R) = g \cdot \text{Res}_x(h, p) \cdot \text{lc}_x(g)^{n'+k'-1} \quad (\text{A.7})$$

Note that $\deg_x(p) = k' > 0$ in induction step, thus we can use first case of Lemma 25 on (A.1) to get

$$\begin{aligned} \text{Res}_x(f, h) &= S_x(0, f, h) \\ &= (-1)^{(\deg_x(f)-0)(\deg_x(h)-0)} \cdot \text{lc}_x(h)^{\deg_x(f)-\deg_x(p)} \cdot S_x(0, h, p) \\ &= (-1)^{m' \cdot n'} \cdot \text{lc}_x(h)^{m'-k'} \cdot \text{Res}_x(h, p) \\ \text{Res}_x(h, p) &= \frac{\text{Res}_x(f, h)}{(-1)^{m' \cdot n'} \cdot \text{lc}_x(h)^{m'-k'}}. \end{aligned} \quad (\text{A.8})$$

Substituting (A.8) in (A.7), we get:

$$S_x(d, B, R) = g \cdot \frac{\text{Res}_x(f, h)}{(-1)^{m' \cdot n'} \cdot \text{lc}_x(h)^{m'-k'}} \cdot \text{lc}_x(g)^{n'+k'-1} \quad (\text{A.9})$$

Substituting (A.9) back into (A.4), we get

$$\begin{aligned} S_x(d, A, B) &= (-1)^{m' \cdot n'} \cdot \text{lc}_x(h)^{m-k} \cdot \text{lc}_x(g)^{m-k} \cdot g \cdot \frac{\text{Res}_x(f, h)}{(-1)^{m' \cdot n'} \cdot \text{lc}_x(h)^{m'-k'}} \cdot \text{lc}_x(g)^{n'+k'-1} \\ &= \text{lc}_x(g)^{m-k} \cdot g \cdot \text{Res}_x(f, h) \cdot \text{lc}_x(g)^{n'+k'-1} \quad [\text{as } m-k = m'-k'] \\ &= g \cdot \text{Res}_x(f, h) \cdot \text{lc}_x(g)^{m-k+n'+k'-1} \\ &= g \cdot \text{Res}_x(f, h) \cdot \text{lc}_x(g)^{m'+n'-1} \quad [\text{as } m-k+k' = m-d = m'] \end{aligned}$$

This completes the proof of induction step, as well as that of the theorem. \blacktriangleleft

A.2 Proof of Lemma 7

Proof of Lemma 7. Without loss of generality, suppose $\sum_{i=1}^n a_i > \sum_{j=1}^m c_j$ and $\sum_{i=1}^n b_i < \sum_{j=1}^m d_j$. In particular, there exists $s \in [n], t \in [m]$ such that $a_s, d_t > 0$. Consider the $2 \times (n+m)$ matrix,

$$E \triangleq \begin{bmatrix} a_1 & \cdots & a_n & c_1 & \cdots & c_m \\ b_1 & \cdots & b_n & d_1 & \cdots & d_m \end{bmatrix}.$$

Suppose E is not full-rank. Since \mathbf{a} and \mathbf{d} are non-zero vectors, E is not the zero matrix and hence it must have rank 1. In that case, the first row is linearly dependent on the second row. Since all E 's entries are non-negative, there exist $\alpha, \beta > 0$ such that $\alpha \cdot \mathbf{a} = \beta \cdot \mathbf{b}$ and $\alpha \cdot \mathbf{c} = \beta \cdot \mathbf{d}$. This implies:

$$\alpha \cdot \left(\sum_{i=1}^n a_i \right) = \beta \cdot \left(\sum_{i=1}^n b_i \right), \quad \alpha \cdot \left(\sum_{j=1}^m c_j \right) = \beta \cdot \left(\sum_{j=1}^m d_j \right).$$

Which in turn implies:

$$\beta \cdot \left(\sum_{i=1}^n b_i \right) = \alpha \cdot \left(\sum_{i=1}^n a_i \right) > \alpha \cdot \left(\sum_{j=1}^m c_j \right) = \beta \cdot \left(\sum_{j=1}^m d_j \right) \implies \sum_{i=1}^n b_i > \sum_{j=1}^m d_j$$

This contradicts the interlacing property. Hence, E must be full-rank. Let E' be a 2×2 rank-2 minor of E . If E' is of the form $E' = \begin{bmatrix} a_i & c_j \\ b_i & d_j \end{bmatrix}$ for some i, j we are done. Otherwise, suppose if E' is of the form $E' = \begin{bmatrix} a_i & a_j \\ b_i & b_j \end{bmatrix}$ or $E' = \begin{bmatrix} c_i & c_j \\ d_i & d_j \end{bmatrix}$ then by the exchange property, we can exchange one of the columns in E' with non-zero columns $\begin{bmatrix} c_t \\ d_t \end{bmatrix}$ or $\begin{bmatrix} a_s \\ b_s \end{bmatrix}$, respectively, to get a rank-2 minor of the required form. \blacktriangleleft

A.3 Proof of Theorem 1

In this section we formally prove our main result - Theorem 1. We begin by restating and proving Corollary 22.

► **Corollary** (Corollary 22). *Let $P, Q \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be products of s -sparse, individual degree- d polynomials such that $P \approx Q$. Let $\mathcal{G} = (\mathcal{G}_1, \dots, \mathcal{G}_n) : \mathbb{F}^t \rightarrow \mathbb{F}^n$ be a generator for $(sd)^{\mathcal{O}(d^3)}$ -sparse, $\mathcal{O}(d^4)$ -individual degree polynomials. Then there exists an $i \in [n]$ such that $P(x_i, \mathcal{G}_{-i}) \approx Q(x_i, \mathcal{G}_{-i})$.*

Proof of Corollary 22. By Lemma 21, we get that $\Phi_{P,Q}(\mathcal{G}) \neq 0$. We deduce that there exists a set $W \subseteq \mathbb{F}$ of large enough size such that $\mathcal{G}(W^t) \subseteq \mathbb{F}^n$ is a hitting set for $\Phi_{P,Q}$. In particular, there exists $\mathbf{b} \in W^t$ such that for $\mathbf{a} \stackrel{\Delta}{=} \mathcal{G}(\mathbf{b})$, we have $\Phi_{P,Q}(\mathbf{a}) \neq 0$. By Lemma 18, there exists an $i \in [n]$ such that $P(x_i, \mathbf{a}_{-i}) \approx Q(x_i, \mathbf{a}_{-i})$. Now suppose $P(x_i, \mathcal{G}_{-i}) \sim Q(x_i, \mathcal{G}_{-i})$. Then have that $P(\mathcal{G}_1(\mathbf{b}), \dots, \mathcal{G}_{i-1}(\mathbf{b}), x_i, \mathcal{G}_{i+1}(\mathbf{b}), \dots, \mathcal{G}_n(\mathbf{b})) \sim Q(\mathcal{G}_1(\mathbf{b}), \dots, \mathcal{G}_{i-1}(\mathbf{b}), x_i, \mathcal{G}_{i+1}(\mathbf{b}), \dots, \mathcal{G}_n(\mathbf{b}))$. This implies that $P(x_i, \mathbf{a}_{-i}) \sim Q(x_i, \mathbf{a}_{-i})$, which is a contradiction. Hence, $P(x_i, \mathcal{G}_{-i}) \approx Q(x_i, \mathcal{G}_{-i})$. \blacktriangleleft

For completeness, we restate Theorem 1.

► **Theorem.** *There exists a deterministic algorithm that given n, d, s and a black-box access to a $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$ circuit C of size s determines if $C \equiv 0$, in time $\text{poly}((sd)^{d^3}, n)$. The algorithm below provides the outline.*

■ **Algorithm 2** Black-box PIT algorithm for class $\Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$.

Input: A polynomial $f(x_1, \dots, x_n) \in \Sigma^{[2]}\Pi\Sigma\Pi^{\text{[ind-deg } d]}$, where bottom $\Sigma\Pi$ computes s -sparse polynomials of individual degrees $\leq d$.

Output: ZERO, if f is identically zero and NON-ZERO, otherwise.

- 1 Invoke [20] to get generator \mathcal{G} of seed-length 1 for n -variate polynomials of sparsity $\leq (sd)^{\mathcal{O}(d^3)}$ and individual degree $\leq \mathcal{O}(d^4)$.
 - 2 **for** $i \leftarrow 1$ **to** n **do**
 - 3 Compute the bivariate polynomial $f(x_i, \mathcal{G}_{-i})$.
 - 4 Do brute-force black-box PIT for $f(x_i, \mathcal{G}_{-i})$.
 - 5 **if** $f(x_i, \mathcal{G}_{-i}) \neq 0$ **then**
 - 6 **return** NON-ZERO.
 - 7 **return** ZERO.
-

10:22 Sparse Polynomial Factorization Related Problems

Proof. We now analyze the correctness and runtime complexity of the Algorithm.

Correctness. Note that $f \equiv 0 \implies f(x_i, \mathcal{G}_{-i}) \equiv 0$ trivially, for all $i \in [n]$. Thus, the algorithm outputs ZERO in this case, as desired. Now suppose $f \not\equiv 0$. Let $f = P + Q$, where both P, Q are product of s -sparse, individual degree d polynomials. If $P \approx Q$, then Corollary 22 implies that there exists an $i \in [n]$ such that $P(x_i, \mathcal{G}_{-i}) \approx Q(x_i, \mathcal{G}_{-i})$. In particular, $P(x_i, \mathcal{G}_{-i}) \neq -Q(x_i, \mathcal{G}_{-i})$. Since $f(x_i, \mathcal{G}_{-i}) = P(x_i, \mathcal{G}_{-i}) + Q(x_i, \mathcal{G}_{-i})$, we deduce that $f(x_i, \mathcal{G}_{-i}) \neq 0$ in this case. Now suppose $f \not\equiv 0$ but $P \sim Q$. Let $P = cQ$, for some $c \in \mathbb{F}$. Then $f = (c + 1)Q$, where $c \neq -1$ and $Q \not\equiv 0$. This means that there exists an $i \in [n]$ such that $\text{lc}_{x_i}(Q) \neq 0$. Using Lemma 21, we know that $\Phi_{P,Q}(\mathcal{G}) \neq 0$ and thus by definition of $\Phi_{P,Q}$, we get $\text{lc}_{x_i}(Q)(\mathcal{G}) \neq 0$. This implies that $Q(x_i, \mathcal{G}_{-i}) \neq 0$. We conclude that $f(x_i, \mathcal{G}_{-i}) = (c + 1)Q(x_i, \mathcal{G}_{-i}) \neq 0$. Thus whenever $f \not\equiv 0$, the algorithm outputs NON-ZERO.

Time complexity. By [20], degree of generator \mathcal{G} is $\text{poly}((sd)^{d^3}, n)$. Note that f has individual degree at most sd and thus $f(x_i, \mathcal{G}_{-i})$ has individual degree $\leq sd \cdot \text{deg}(\mathcal{G})$. Testing non-zeroness of the bivariate polynomial $f(x_i, \mathcal{G}_{-i})$ takes only $\text{poly}((sd)^{d^3}, n)$ time. The n iterations only add a factor of n . ◀

Complexity of Spatial Games

Krishnendu Chatterjee

Institute of Science and Technology Austria, Klosterneuburg, Austria

Rasmus Ibsen-Jensen

University of Liverpool, UK

Ismaël Jecker

University of Warsaw, Poland

Jakub Svoboda

Institute of Science and Technology Austria, Klosterneuburg, Austria

Abstract

Spatial games form a widely-studied class of games from biology and physics modeling the evolution of social behavior. Formally, such a game is defined by a square (d by d) payoff matrix M and an undirected graph G . Each vertex of G represents an individual, that initially follows some strategy $i \in \{1, 2, \dots, d\}$. In each round of the game, every individual plays the matrix game with each of its neighbors: An individual following strategy i meeting a neighbor following strategy j receives a payoff equal to the entry (i, j) of M . Then, each individual updates its strategy to its neighbors' strategy with the highest sum of payoffs, and the next round starts. The basic computational problems consist of reachability between configurations and the average frequency of a strategy. For general spatial games and graphs, these problems are in PSPACE. In this paper, we examine restricted setting: the game is a prisoner's dilemma; and G is a subgraph of grid. We prove that basic computational problems for spatial games with prisoner's dilemma on a subgraph of a grid are PSPACE-hard.

2012 ACM Subject Classification Theory of computation

Keywords and phrases spatial games, computational complexity, prisoner's dilemma, dynamical systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.11

Funding *Krishnendu Chatterjee*: The research was partially supported by the ERC CoG 863818 (ForM-SMArt).

Ismaël Jecker: The research was partially supported by the ERC grant 950398 (INFSYS).

Jakub Svoboda: The research was partially supported by the ERC CoG 863818 (ForM-SMArt).

1 Introduction

Spatial evolutionary games is a classic and well-studied model of evolutionary dynamics on graphs, which has been studied across fields, e.g., biology [9, 8], physics [10, 14], and computer science [3, 2].

While computer science studies games with few players and a large number of actions, evolutionary game theory studies games with few actions and strategies but with many players (see the survey [17]). Specifically, each spatial evolutionary game consists of a square, skew-symmetric, bimatrix game (i.e. the outcome in entry (i, j) for player 1 is the same as the outcome for player 2 in (j, i) for all i, j) and a finite graph. The game is played over a number of rounds. Each node of the graph corresponds to a player. Each node/player is associated with a current row and corresponding column. In each round, each player plays the matrix game against each of their neighbors, by playing their row against their neighbor's column (because of the skew-symmetry, who plays rows and who plays columns does not matter) and gets a payoff assigned, which is the sum of outcomes of the games they played



© Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Ismaël Jecker, and Jakub Svoboda;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 11; pp. 11:1–11:14



Leibniz International Proceedings in Informatics

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

11:2 Complexity of Spatial Games

in that round. Each player then switches to the row played by their neighbor that had the highest payoff (or keeps their strategy). Since this is a deterministic dynamic, whenever we reach a round such that there is a previous round in which each player had the same row as now, the game “loops”. All spatial evolutionary games will therefore loop after at most d^n rounds when the bi-matrix is d by d and there are n nodes.

Most studied setup: Grids and prisoner’s dilemma. The standard study of spatial evolutionary games focuses on small (2 by 2 or 3 by 3) bi-matrices and on grids. A good understanding of the dynamics is known for a number of such setups (including all 2 by 2 bi-matrices on 2-dimensional grids). Grids are typically chosen since in biology they naturally model how cells interact with nearby cells. In particular, the focus has been on prisoner’s dilemma (PD) matrices: A prisoner’s dilemma bi-matrix is a 2 by 2 bi-matrix in which the two rows are called cooperation and defection, such that it is an advantage to defect if the other player cooperates, but it is better for both players if they both cooperate as compared to the mutual defection. We study these games to better understand why cooperation develops as we see in humans and many animal species. Indeed, this was the focus of the original paper [8] and later works extended this basic model in myriad ways:

1. What happens if you add in mobile agents [19, 5]?
2. What about age [11, 21]?
3. What if nodes/players do not have the same objectives [18, 13]?
4. What if there were different timescales [14, 22]?

See also the survey [12]. A common generalization considers more general graph types. We mention a few examples of the papers pursuing this direction:

1. Kabir et al. showed how increasing the network reciprocity changes the likelihood of cooperation in PD [6].
2. Hassell et al. considered how multiple different species on models with islands influence the outcome [4].
3. Santos and Pacheco showed how scale-free networks (when generated following specific paradigms) promote cooperation [15].
4. Yamauchi et al. showed how, if both the neighbors and strategies can change over time, cooperation can evolve [16].
5. Ohtsuki et al. gave a simple heuristic for when cooperation can evolve on a variety of different networks, including social networks [9].

Computational problems. The works mentioned previously usually examine how cooperation spreads when the process is applied many times. The question they are trying to decide is: Given a starting position, what is the average number of cooperators in the long run?

Open questions. The general spatial games problem is in PSPACE. This is because a configuration consists of a strategy for every vertex, which can be stored in polynomial space, and the update of the configuration according to the rules can also be achieved in PSPACE. The most well-studied problem for spatial games is the prisoner’s dilemma, which has only two strategies, namely, cooperation and defection. Moreover, such games have been studied for special classes of graphs. The main open question is whether efficient algorithms can be obtained for prisoner’s dilemma on graphs like grids.

Our results. In this paper, we consider spatial evolutionary games with a prisoner’s dilemma matrix on *subsets*¹ of 2-dimensional grids. The subset models the situation when locations of e.g. cells or connections between them have been destroyed or are otherwise inaccessible.

Our main result is that for subgraphs of a two-dimensional grid and prisoners dilemma the reachability and average cooperation problems are PSPACE-hard. Additionally, we show that induced subgraphs can loop in loops of exponential length. Subsets of grids are simpler than scale-free or evolving networks, so our hardness result holds for more general graphs.

2 Model and definitions

Graphs and grids. A graph $G = (V, E)$ consists of a set of vertices V and a set of undirected edges E . Every vertex is occupied by one individual. Edges determine pairs of individuals that interact. Two-dimensional grids are specific graphs where each vertex is assigned a unique pair of integers (i, j) , and has (at most) 8 neighbors: vertices whose pairs differ by at most 1 in both coordinates. In our construction, we use graphs derived from grids: Given a grid (V, E) , a *induced subgraph* of (V, E) is a graph $(V', (V' \times V') \cap E)$ with $V' \subseteq V$. An *subgraph* of (V, E) is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$.

Games and individuals. An individual occupying a vertex has one of two types: cooperator if it plays C or defector if it plays D . In the figures, we use black for cooperators and white for defectors. We call *configuration* of a graph an assignment of each vertex to a strategy (cooperator or defector).

From all matrix games, we focus on prisoner’s dilemma. The game is denoted by a matrix

	C	D
C	1	0
D	b	0

where $b > 1$. It means that C gets 1 for interacting with C , D gets b for interacting with C , and everyone gets 0 for interacting with D . We denote this game M^b .

Steps and updates. The evolution is simulated in rounds. In one round, every individual interacts with all neighbors and collects the total payoff. Then every individual compares the received payoff with the payoffs of neighbors. The individual keeps the strategy if it is the highest or changes the strategy to the strategy of a neighbor with the highest payoff (in a tie, defection is preferred).

We denote the process starting from position S on graph G with a game matrix M^b as $\mathcal{S}(G, M^b, S)$.

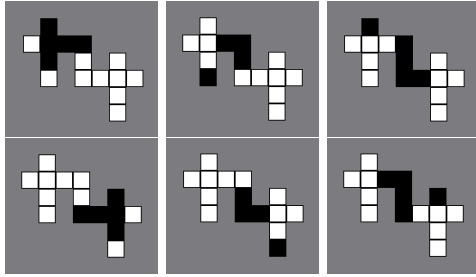
Complexity problems. We consider two complexity problems.

REACH : Given starting configuration, does the process reach a given configuration?

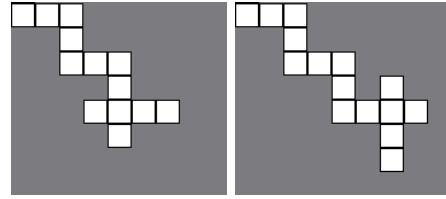
AVG : For a given starting configuration, what is the average number of cooperators (black vertices) in all succeeding configurations?

Note that for one configuration, there is only one possible succeeding configuration. This restricts the configuration graph. That means eventually the configuration graph creates a loop (as was noted before [20]).

¹ Either subgraphs or induced subgraphs



■ **Figure 1** All configurations of the gadget c_3 initiated by the top left configuration. These configurations (in rows) show period 6.



■ **Figure 2** Extension of the gadget c_k by two cells. We extend the gadget to the lower right, the gadget itself can be as long as needed connected to the upper left which increases period by 2.

Ranges of b . There is a reasonable range for b in M^b . If $b < 1$, then the game is not a prisoner’s dilemma. If b is larger than the maximal degree in a graph, the dynamic is trivial, a defector cannot become a cooperator. For our constructions in this paper, we suppose that $b \in (\frac{3}{2}, 2)$. In Appendix A, we show ideas explaining how our construction can be adapted to other values of b .

3 Exponential cycle

In this section, we show that even on an induced subgraph of a square grid, we observe a complex behavior. Namely, there exists a graph and a configuration that returns back to the starting configuration only after an exponential number of steps, we use $\tilde{\Omega}$ and $\tilde{\mathcal{O}}$ which hide logarithmic factors.

► **Theorem 1.** For $b \in (\frac{3}{2}, 2)$, there exists a graph G , induced subgraph of a square grid, with n vertices and a starting configuration S , such that it takes $2^{\tilde{\Omega}(\sqrt{n})}$ steps until $S(G, M^b, S)$ reaches S again.

Proof. We describe a family of gadgets and starting configurations with different periods, where *period* is the number of steps the gadget needs to return to the starting position again. Then we combine some of them to create a graph with a period equal to the lowest common multiple of the periods of its components.

For $k \geq 3$, we construct inductively c_k , a gadget that is an induced subgraph of a square grid with size $9 + 2k$ and period $2k$. The gadget c_3 in its starting configuration is depicted on Figure 1. By rearranging and adding two squares, we create the gadget c_{k+1} from c_k , as depicted on Figure 2.

For every integer $g > 1$, let us now define G_g as the disjoint union of the gadgets $c_{p_2}, c_{p_3}, \dots, c_{p_g}$ where p_i is the i -th prime number. By the Chinese remainder theorem, the number of steps needed so that all gadgets are in the same state is the smallest common multiple of their periods, which is the product of the first g primes. We know that this number is $\Theta(e^{g \log g})$ from [7] and the Prime Number Theorem. Moreover, the number of squares (vertices) of G_g linear in the sum of the first g primes which is $\tilde{\mathcal{O}}(g^2)$.

Therefore, by using an induced subgraph of the square grid of size $n \in \mathbb{N}$, we can create a union of gadgets that has a period of size $2^{\tilde{\Omega}(\sqrt{n})}$. ◀

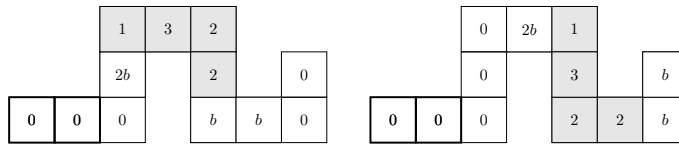


Figure 3 Sending signal through the wire with explicit payoffs with cooperators denoted by gray. Thicker vertices are input vertices.

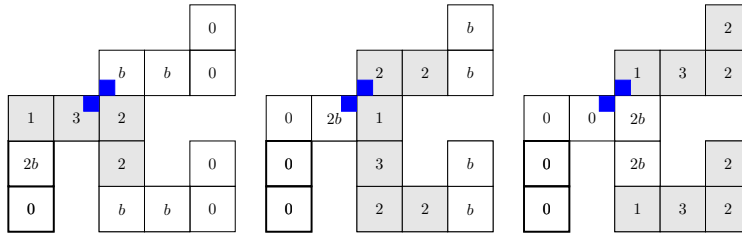


Figure 4 Splitting the signal in two. Blue boxes denote a deleted edge with cooperators denoted in black. Thicker vertices are input vertices.

4 Construction of a Turing machine

We show that both REACH and AVG are P-SPACE hard. For a polynomially bounded Turing machine and its input, we create a graph of a polynomial-size that is a subgraph of the square grid and an initial configuration of cooperators and defectors such that the process reaches a predefined configuration if and only if the Turing machine accepts the given input.

Since both problems, REACH and AVG, can be easily solved in P-SPACE by a simulation, that means these problems are P-SPACE complete.

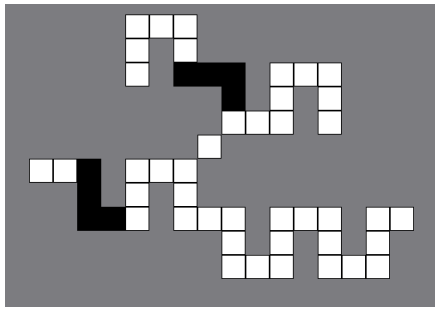
► **Theorem 2.** For $b \in (\frac{3}{2}, 2)$ holds:

For any Turing machine T polynomially bounded by n and its input, there exists a subgraph of a square grid G with $\text{poly}(n)$ vertices, a starting configuration S , and a target configuration Q , such that $\mathcal{S}(G, M^b, S)$ reaches Q if and only if T accepts the given input.

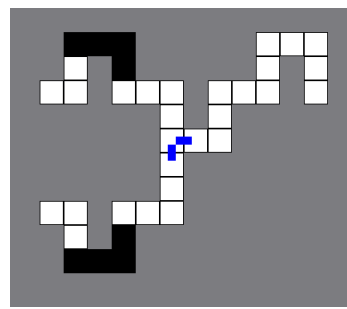
Moreover, for any Turing machine T and its input I , there exists a subgraph of an infinite square grid G , a starting configuration S with $\text{poly}(|I|)$ cooperators, and a target configuration Q , such that $\mathcal{S}(G, M^b, S)$ reaches Q if and only if T accepts the given input.

We construct G as a white (defector) graph with a few black (cooperator) vertices that carry signals and store data to simulate the behavior of T . On our figures, we use white for defectors, black for cooperators, gray for deleted vertices, and blue for *deleted edges*: To visualize deleted edges, we subdivide each square denoting some vertex v into 9 parts. The middle square corresponds to v itself, and the other squares denote the neighbors in relative position to v (upper-left, upper-middle, ...). To represent that an edge between two vertices v and w is deleted, we color in blue the subsquare corresponding to w in v 's square and the subsquare corresponding to v in w 's square.

First, we describe wires and basic logic gates. With that, we use a construction described in [1]. We use Lemma 7 and 9 from the paper, but explain technicalities emerging from more restrictive construction (the graph is a subgraph of a square grid). Both lemmas use simple gadgets to create functions and then a whole Turing machine.



■ **Figure 5** NOT gate: the upper signal is negated, the lower signal is a clock signal.



■ **Figure 6** AND gate: both signals are essential to tunnel through. There are two deleted edges denoted by blue boxes.

4.1 Basic gadgets

We describe the computational gadgets used in our construction. Every gadget g has one or two inputs (I_1, I_2) and one or two outputs (O_1, O_2). We imagine the inputs being on the left and bottom and the outputs on the right and top. Every gadget fits into a constantly sized rectangle of the grid.

If at time t at least one input of g is true (input vertices are cooperators and other vertices are defectors), then at time $t + t_g$ it outputs true or false based on a function g computes.

One input (or output) consists of two connected vertices. We say that the input is true if both vertices are cooperators at time t and $t + 1$, and in the first step, they can convert all neighbors to cooperators. The previous gadget (connected by its output to the given input) is responsible for turning the input vertices back to defectors at time $t + 2$.

Here, some of the gadgets need to cross signals, we describe how to do it later:

Wire: it transmits a signal (See Figure 3). The wire has one input, one output, and if I_1 is true at time t , then O_1 is true at time $t + 2$.

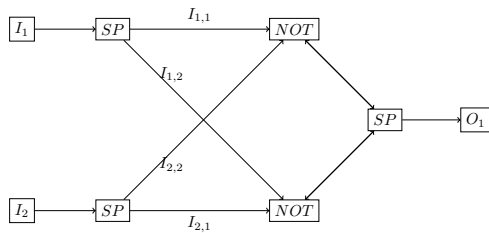
Splitter: it splits one signal in two (See Figure 4). The splitter has one input and two outputs. If I_1 is true at time T , then O_1 and O_2 are true at time $t + 4$. An interesting property of the splitter is that it does not matter if the signal arrives from I_1 or O_1 . From the graph perspective, these two are symmetric.

NOT gate: it computes the logical negation, but necessitates a clock signal (See Figure 5). The NOT gate has two inputs and one output. Input I_2 is a clock input: if I_1 and I_2 are true at time t , then O_1 is false at time $t + 8$, if only I_2 is true at time t , then O_1 is true at time $t + 8$. Note that the NOT gate actually computes the function $\neg I_1 \text{ AND } I_2$, also the gates I_1 and O_1 are interchangeable.

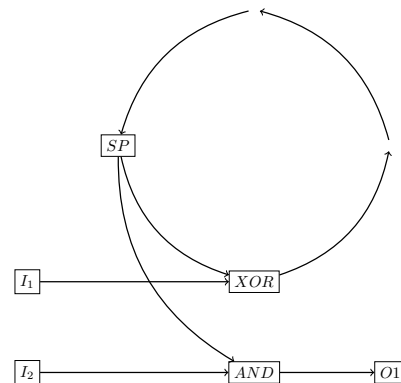
AND gate: it computes the logical conjunction (See Figure 6). The AND gate has two inputs and one output. If both I_1 and I_2 are true at time t , then O_1 is true at time $t + 8$. Otherwise, O_1 is false.

XOR gate: it computes logical exclusive disjunction. We can create it from other gadgets, but it makes our construction easier.

We compute XOR using connection of splitter and NOT gates (see Figure 7). It has two inputs I_1 and I_2 and one output O_1 . First every input is split to get $I_{1,1}, I_{1,2}, I_{2,1}$ and $I_{2,2}$, then $I_{1,1}$ is sent to I_1 of NOT gate and $I_{2,1}$ is sent to O_1 of a splitter S . We already know that if only one signal equals true, then S splits the signal. If both are positive, signals annihilate each other. So O_2 of S has value $I_1 \text{ XOR } I_2$. The only problem is when



■ **Figure 7** XOR gate: Combination of previous gates. Signals $I_{1,2}$ and $I_{2,2}$ are delayed such that they arrive at the NOT gate when signal $I_{1,1}$ or $I_{2,1}$ from the central splitter would.



■ **Figure 8** Storage unit.

I_1 is true, and I_2 is not (or symmetric). Then the signal travels back from the splitter through wire $I_{2,1}$, then we use the NOT gate with clock signal $I_{2,1}$ and signal $I_{1,2}$ that stops it.

OR gate: it computes logical disjunction. It has two inputs I_1 and I_2 , and the output O_1 satisfies the function $(I_1 \text{ AND } I_2) \text{ XOR } (I_1 \text{ XOR } I_2)$, it consists of gadgets described above. Note that we don't need the NOT gate directly, so clock signal is not necessary for that gadget.

Another gadget that the construction needs is a storage unit. It has an inner state $S \in \{0, 1\}$, two inputs and one output, see Figure 8. The storage unit consists of a big cyclic wire where a signal loops if the stored value S is 1. If a signal is sent via I_1 , then the storage unit changes state: $S' = \neg S$, this is ensured by a XOR gate. On the cycle, there is a splitter that splits the signal towards an AND gate. If a signal is sent via I_2 , S does not change, but the signal reaches AND and is sent to O_2 if and only if $S = 1$. The storage unit requires synchronicity, the signal from the input has to reach the splitter or XOR at the right time. But this is not hard to ensure by longer wires.

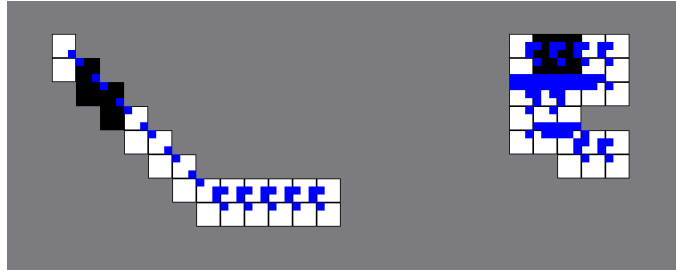
4.2 Connecting the graph

To make the graph a subgraph of a square grid, we need to prove two things: we can cross two signals, and we can ensure that the signals meet at the right place, at the right time.

Crossing. We use the structure described in Figure 13. Crossing accepts only one signal at a time and supposes that no other signal arrives for 10 steps afterward. Crossing ensures that in a square subset of a grid, the signal can travel only from upper left to lower right corner and from lower left to upper right without spreading or dying out. If a signal arrives, it is spreading to the other input and all the outputs, but the wires close to the active input get stopped, so only the wire that is not adjacent to the input wire continues to carry the signal.

Making the construction on square grid. On the Figure 9, we see that a signal starting at position $(0, 0)$ going to position (x, y) can take between $\max(x, y)$ and $\frac{1}{8} \cdot xy$ steps. The signal also does not leave the rectangle denoted by points $(0, 0)$ and (x, y) .

Every gadget that was described above can be padded by wires of different densities such that the gadget fits into a rectangle with predetermined width and height. Moreover, all inputs are in the same position (relative to the rectangle) and the time of evaluation is the same for all (constant).



■ **Figure 9** Two wires of different lengths connecting two points.

Then everything in the following construction can be viewed as placing a column of tiles (gadgets) one after another, where columns are connected by short wires.

4.3 Function construction

Here we construct a function using previously described gadgets. We bound the number of vertices and steps needed for the construction and the function evaluation.

We imagine a function as signals going from left to right. The input and output signals have constant horizontal distance.

► **Definition 3.** We say that a function $f : \{0, 1\}^k \rightarrow \{0, 1\}^l$ is computed by a graph G (subgraph of a grid) in time g and space h if G has h vertices, k inputs I_1, I_2, \dots, I_k and l outputs O_1, O_2, \dots, O_l spatially arranged, and such that for all $(x_1, x_2, \dots, x_k) \in \{0, 1\}^k$, if at the time t we have $I_j = x_j$ for all $1 \leq j \leq k$ (and all the other vertices are defectors), then at the time $t + g$ we have $O_j = y_j$ for all $1 \leq j \leq l$, where $f(x_1, x_2, \dots, x_k) = (y_1, y_2, \dots, y_l)$.

Note that all our basic gadgets have a constant size. So, when analyzing the asymptotic space complexity, we can consider these gadgets and single vertices interchangeably.

► **Lemma 4.** Computing the function $f(x_1, x_2, \dots, x_c) = (x_1, x_2, \dots, x_c, x_1, x_2, \dots, x_c)$ takes time $\mathcal{O}(c)$ and space $\mathcal{O}(c^2)$.

Proof. We describe the gadget computing f . First, we split every signal and then cross every copy of it with others to the right place. One signal needs to cross $\mathcal{O}(c)$ others, so that is the number of steps and we have c signals going through a wire of length c , that makes $\mathcal{O}(c^2)$ gadgets. ◀

► **Lemma 5.** Let $c \in \mathbb{N}$. Every function $f : \{0, 1\}^c \rightarrow \{0, 1\}$ mapping to 0 every tuple whose first component is 0 can be computed in space $\mathcal{O}(3^c)$ and time $\mathcal{O}(c^2)$.

Proof. We use an induction on the dimension c of the domain. If $c = 1$, since f satisfies $f(0) = 0$ by supposition, either f is the identity, which is realised by a wire, or f is the zero function, which is realised by simply disconnecting the input and output.

Now, suppose that $c > 1$, and we can realise any function $f : \{0, 1\}^{c-1} \rightarrow \{0, 1\}$ that satisfies the condition. In particular, there exist two gadgets g_0 and g_1 computing

$$\begin{aligned} f_0 : \quad & \{0, 1\}^{c-1} && \rightarrow && \{0, 1\}, \\ & (x_1, x_2, \dots, x_{c-1}) && \mapsto && f(x_1, x_2, \dots, x_{c-1}, 0), \\ f_1 : \quad & \{0, 1\}^{c-1} && \rightarrow && \{0, 1\}, \\ & (x_1, x_2, \dots, x_{c-1}) && \mapsto && f(x_1, x_2, \dots, x_{c-1}, 1). \end{aligned}$$

Having these values, the computation is straightforward as the value $f(x_1, x_2, \dots, x_c)$ is given by the formula

$$(f_0(x_1, x_2, \dots, x_{c-1}) \wedge \neg x_c) \vee (f_1(x_1, x_2, \dots, x_{c-1}) \wedge x_c).$$

To construct a gadget g that is a subgraph of a grid g and computes f , we proceed as follows.

Computing the function. We apply the gadget described in Lemma 4 to the input, and we use a splitter to get one more signal x_1 . At this point, we have the arranged signals $x_1, x_1, x_2, x_3, \dots, x_c, x_1, x_2, x_3, \dots, x_c$. We apply the gadgets computing f_0 and f_1 to get the signals $f_0(x_1, x_2, \dots, x_c)$ and $f_1(x_1, x_2, \dots, x_c)$, now the signals are arranged as $x_1, f_0(x_1, x_2, \dots, x_{c-1}), x_c, f_1(x_1, x_2, \dots, x_{c-1}), x_c$. We cross the signals x_1 and $f_0(x_1, x_2, \dots, x_{c-1})$, and then we use x_1 as clock signal towards a NOT gate with x_c . By this, we either get the signal $\neg x_c$, or x_1 is zero, then the result should be zero anyway.

Now we send $f_0(x_1, x_2, \dots, x_{c-1})$ and $\neg x_c$ towards an AND gate, similarly we send $f_1(x_1, x_2, \dots, x_{c-1})$ and x_c towards another AND gate, and finally we send both results towards an OR gate.

Size of the gadget. Now, we show that the computation is fast and requires a reasonable number of vertices. Let there be a recurrent formula R_s that maps any integer c to the maximal number of gadgets needed to compute the function f . Using induction, we see that it satisfies:

$$R_s(c) = \mathcal{O}(c^2) + 2R_s(c-1) + \mathcal{O}(1).$$

Therefore, $R(c) \in \mathcal{O}(3^c)$. Similarly, we use recurrent formula for the time needed

$$R_t(c) = \mathcal{O}(c) + R_t(c-1) + \mathcal{O}(1)$$

and the solution for this recurrence is $\mathcal{O}(c^2)$. ◀

► **Lemma 6.** *Let $c, d \in \mathbb{N}$. Every function $f : \{0, 1\}^c \rightarrow \{0, 1\}^d$ mapping to $(0, 0, \dots)$ every tuple whose first component is 0 can be computed in space $\mathcal{O}(d3^c)$ and time $\mathcal{O}(c^2 + c \log(d))$.*

Proof. The idea is easy, we split the inputs d times using Lemma 4 and then we use Lemma 5 for every copy.

Multiplying the inputs needs $\mathcal{O}(d)$ described in Lemma 4. We can arrange them in layers where every layer doubles the input, so the whole multiplying takes time $\mathcal{O}(\log(d) \cdot c)$.

Then we use d gadgets described in Lemma 5 in parallel which gives time $\mathcal{O}(c^2 + c \log(d))$ and space $\mathcal{O}(d3^c)$. ◀

4.4 Blob and connections

► **Lemma 7.** *Let T be a Turing machine. For every input u evaluated by T using $C \in \mathbb{N}$ cells of the tape, there exists a subgraph of a grid G on $\mathcal{O}(C)$ vertices and an initial configuration c_0 of G such that T stops over the input u if and only if $\mathcal{S}(G, M^b, c_0)$ for $b \in (\frac{3}{2}, 2)$ eventually reaches a configuration without cooperators.*

Proof. We suppose that the Turing machine T has a single final state, which can only be accessed after clearing the tape. We present the construction of the graph G simulating T through the following steps. First, we encode the states of T , the tape alphabet, and the transition function in binary. Then, we introduce the notion of a blob, the building block of G , and we show that blobs accurately simulate the transition function of T . Afterward, we approximate the size of a blob, and finally, we define G as a composition of blobs.

11:10 Complexity of Spatial Games

Binary encoding. Let $T_s \in \mathbb{N}$ be the number of states of T , and $T_a \in \mathbb{N}$ be the size of its tape alphabet. We pick two small integers s and n satisfying $T_s \leq 2^{s-1}$ and $T_a \leq 2^{n-1}$. We encode the states of T as elements of $\{0, 1\}^s$, and the alphabet symbols as elements of $\{0, 1\}^n$, while respecting the following three conditions: the blank symbol maps to 0^n , the final state of T maps to 0^s , and all the others map to strings starting with 1. Then, for these mappings, we modify the transition function of T to:

$$F : \{0, 1\}^s \times \{0, 1\}^n \rightarrow \{0, 1\}^s \times \{0, 1\}^s \times \{0, 1\}^n.$$

Instead of using one bit to denote if the head is going left or right, we use $2s$ bits to store the state and signify the movement: if the first s bits are zero, the head is moving right; if the second s bits are zero, it is moving left; if the first $2s$ bits are zero, the computation ended. Moreover, the last n bits of the image of F do not encode the new symbol, but the symmetric difference between the previous and the next symbol: if the i -th bit of the tape symbol goes from y_i to z_i , then F outputs $d_i = y_i \oplus z_i$ (XOR of these two).

Constructing blobs. We construct the graph G by simulating each cell of the tape with a blob. Blob stores a tape symbol, and after receiving a signal corresponding to a state of T it computes the transition function. The main components of a blob are as follows.

- Memory: n storage units (m_1, m_2, \dots, m_n) are used to keep in memory a tape symbol $a \in \{0, 1\}^n$ of T .
- Receptor: $2s$ inputs $(I_1, I_2, \dots, I_{2s})$ are used to receive states $q \in \{0, 1\}^s$ of T either from the left or from the right.
- Transmitter: $2s$ outputs $(O_1, O_2, \dots, O_{2s})$ are used to send states $q \in \{0, 1\}^s$ of T either to the right or to the left.
- Transition gadget: We use gadget from Lemma 6, it needs $\mathcal{O}((n+s)3^{n+s})$ space and $\mathcal{O}((n+s)^2)$ time.

Blobs are connected in a row to act as a tape: for every $1 \leq i \leq s$, the output O_i of each blob connects to the input I_i of the blob to its right, and the output O_{s+i} of each blob connects to the input I_{s+i} of the blob to its left. When receiving a signal, the blob transmits the received state and the tape symbol stored in memory to the transition gadget g_F , which computes the corresponding transition, and then apply its results. We now detail this inner behavior. Note that when a gadget is supposed to receive simultaneously a set of signals coming from different sources, it is always possible to add wires of adapted length to ensure that all of them end up synchronized.

Simulating the transition function. To simulate the transition function of T , a blob acts according to the three following steps:

1. Transmission of the state. A blob can receive a state either from the left (through inputs I_1, I_2, \dots, I_s) or from the right (through inputs $I_{s+1}, I_{s+2}, \dots, I_{2s}$), but not from both sides at the same time, since at every point in time there is at most one active state. Therefore, if for every $1 \leq i \leq s$ we denote by x_i the disjunction of the signals received by I_i and I_{s+i} , then the resulting tuple (x_1, x_2, \dots, x_s) is equal to the state received as signal (either from the left or the right), which can be fed to the gadget g_F . Formally, the blob connects, for all $1 \leq i \leq s$, the pair I_i, I_{s+i} to an OR gate whose output is linked to the input I_i of g_F .
2. Transmission of the tape symbol. Since the first component of any state apart from the final state is always 1, whenever a blob receives a state, the component x_1 defined in the previous paragraph has value 1. The tape symbol (y_1, y_2, \dots, y_n) currently stored in the

blob can be obtained by sending, for every $1 \leq i \leq n$, a copy of x_1 to the input I_2 of the storage unit s_i , causing it to broadcast its stored state y_i . The tuple continues to the gadget g_F . Formally, the blob uses n splitters to transmit the result of the OR gate between I_1 and I_{s+1} to the input I_1 of each storage unit. Then, for every $1 \leq i \leq n$, the output O_1 of the storage unit s_i is connected to the input I_{s+i} of g_F .

3. Application of the transition. Upon receiving a state and a tape symbol, g_F computes the result of the transition function, yielding a tuple $(r_1, r_2, \dots, r_{s+n})$. The blob now needs to do two things: send a state to the successor blob and update the element of the tape.

Connecting the output O_i of g_F to the output O_i of the blob for every $1 \leq i \leq 2s$ ensures that the state is sent to the correct neighbor: the values (r_1, r_2, \dots, r_s) are nonzero if the head is supposed to move to the next block on the right (outputs O_1, O_2, \dots, O_s are connected that blob). Conversely, $(r_{s+1}, r_{s+2}, \dots, r_{2s})$ are nonzero if the head is supposed to move to the left, (outputs $O_{s+1}, O_{s+2}, \dots, O_{2s}$ are connected to the left blob).

Finally, connecting the output O_{2s+i} of g_F to the input I_1 of s_i for all $1 \leq i \leq n$ ensures that the state is correctly updated: this sends the signal d_i to the input I_1 of the storage unit s_i . Since d_i is the difference between the current bit and the next, the state of s_i will change if it has to.

The size of blob. The size of the blob is determined by the size of the transition gadget g_F : one blob is composed of $\mathcal{O}(3^{n+s})$ vertices, and evaluating a transition requires $\mathcal{O}((n+s)^2)$ steps by Lemma 6. Since n and s are constants (they depend on T , and not on the input u), the blob has constant size. Crossing wires to get them to the right place also takes space $\mathcal{O}((n+s)^2)$.

Constructing G . Now that we have blobs that accurately simulate the transition function of T , constructing the graph G mimicking the behavior of T over the input u is straightforward: we take a row of C blobs (remember that $C \in \mathbb{N}$ is the number of tape cells used by T to process u). Since the size of a blob is constant, the size of G is polynomial in C . We define the initial configuration of G by setting the states of the $|u|$ blobs on the left of the row to the letters of u , and setting the inputs I_1 to I_s of the leftmost blob to the signal corresponding to the initial state of T as if it was already in the process. As explained earlier, the blobs then evolve by following the run of T . If the Turing machine stops, then its tape is empty, the final state is encoded by 0^s , and the blank symbol is encoded by 0^n . So G reaches the configuration where all vertices are defectors. Conversely, if T runs forever starting from the input u , there will always be some cooperators vertices in G to transmit the signal corresponding to the state of T . ◀

Proof of Theorem 2, part 1. By Lemma 7, we can reduce any problem solvable by a polynomially bounded Turing machine into REACH, asking whether we reach the configuration with only defectors. Or into AVG, asking whether the long-run average is strictly above 0. ◀

Proof of Theorem 2, part 2. Again, by Lemma 7, we can create an infinite chain of blobs and then we can reduce any problem solvable by any Turing machine into REACH, asking whether we reach the configuration with only defectors, or into AVG, asking whether the long-run average is strictly above 0. ◀

5 Conclusion

Our result shows that going beyond the basic grid model even slightly makes spatial games PSPACE-hard. It ensures that there is no efficient (polynomial-time) algorithm for REACH or AVG, even for the simplest game of prisoner's dilemma.

We studied games with synchronous and deterministic updating. It poses a question: does lifting any restriction permit an efficient algorithm? Moreover, we can ask whether it is possible to construct graphs on which the game has certain properties, such as: are there graphs where the cooperative behavior is favored?

References

- 1 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Ismaël Jecker, and Jakub Svoboda. Simplified game of life: Algorithms and complexity. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPICs*, pages 22:1–22:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.MFCS.2020.22.
- 2 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009.
- 3 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.
- 4 Michael P. Hassell, Hugh N. Comins, and Robert M. May. Species coexistence and self-organizing spatial dynamics. *Nature*, 370:290–292, 1994.
- 5 Dirk Helbing and Wenjian Yu. The outbreak of cooperation among success-driven individuals under noisy conditions. *Proceedings of the National Academy of Sciences*, 106(10):3680–3685, 2009.
- 6 KM Ariful Kabir, Jun Tanimoto, and Zhen Wang. Influence of bolstering network reciprocity in the evolutionary spatial prisoner's dilemma game: a perspective. *The European Physical Journal B*, 91(12), December 2018.
- 7 Sebastián M. Ruiz. 81.27 a result on prime numbers. *The Mathematical Gazette*, 81:269, July 1997. doi:10.2307/3619207.
- 8 Martin A. Nowak and Robert M. May. Evolutionary games and spatial chaos. *Nature*, 359(6398):826–829, October 1992. doi:10.1038/359826a0.
- 9 Hisashi Ohtsuki, Christoph Hauert, Erez Lieberman, and Martin A. Nowak. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441(7092):502–505, May 2006. doi:10.1038/nature04605.
- 10 Hisashi Ohtsuki, Martin A. Nowak, and Jorge M. Pacheco. Breaking the symmetry between interaction and replacement in evolutionary dynamics on graphs. *Phys. Rev. Lett.*, 98:108106, March 2007. doi:10.1103/PhysRevLett.98.108106.
- 11 Matjaž Perc and Attila Szolnoki. Social diversity and promotion of cooperation in the spatial prisoner's dilemma game. *Phys. Rev. E*, 77:011904, January 2008.
- 12 Matjaž Perc and Attila Szolnoki. Coevolutionary games—a mini review. *Biosystems*, 99(2):109–125, 2010.
- 13 Matjaž Perc and Zhen Wang. Heterogeneous aspirations promote cooperation in the prisoner's dilemma game. *PLOS ONE*, 5(12):1–8, December 2010.
- 14 Carlos P. Roca, José A. Cuesta, and Angel Sánchez. Time scales in evolutionary dynamics. *Phys. Rev. Lett.*, 97:158701, October 2006.
- 15 Francisco C. Santos and Jorge M. Pacheco. Scale-free networks provide a unifying framework for the emergence of cooperation. *Phys. Rev. Lett.*, 95, August 2005.
- 16 Francisco C. Santos, Jorge M. Pacheco, and Tom Lenaerts. Cooperation prevails when individuals adjust their social ties. *PLOS Computational Biology*, 2(10):1–8, October 2006. doi:10.1371/journal.pcbi.0020140.

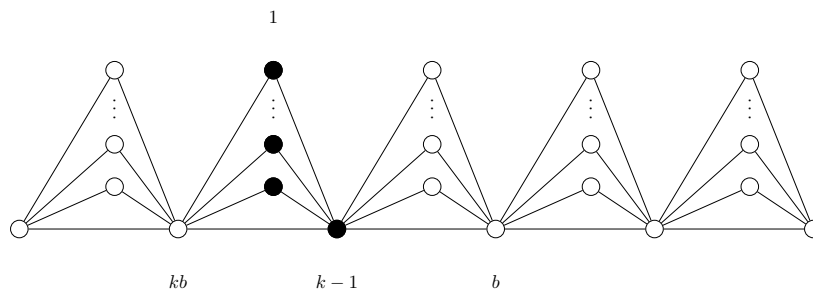
- 17 Gyorgy Szabo and Gabor Fath. Evolutionary games on graphs. *Physics Reports*, 446(4):97–216, 2007.
- 18 Attila Szolnoki and Gyorgy Szabó. Cooperation enhanced by inhomogeneous activity of teaching for evolutionary prisoner's dilemma games. *Europhysics Letters (EPL)*, 77(3):30004, January 2007. doi:10.1209/0295-5075/77/30004.
- 19 Mendeli H. Vainstein, Ana T.C. Silva, and Jeferson J. Arenzon. Does mobility decrease cooperation? *Journal of Theoretical Biology*, 244(4):722–728, 2007.
- 20 Virgil. *Eclogue IV*. Cambridge University Press, 2008.
- 21 Zhen Wang, Zhen Wang, Xiaodan Zhu, and Jeferson J. Arenzon. Cooperation and age structure in spatial games. *Phys. Rev. E*, 85:011149, January 2012.
- 22 Zhi-Xi Wu, Zhihai Rong, and Petter Holme. Diversity of reproduction time scale promotes cooperation in spatial prisoner's dilemma games. *Phys. Rev. E*, 80:036106, September 2009.

A Construction for a more general b

We can make a similar construction for different b 's than those from the range $(\frac{3}{2}, 2)$. By selecting different b , the construction is not a subgraph of a square grid.

We can interpret the wire from Figure 3 as a path (lower row of vertices) with auxiliary vertices (upper row) that empower the tip of the cooperator signal and also help the first defector behind it. For different b 's, we can add more auxiliary vertices. Details are on Figure 10.

For setting $b \in (\frac{3}{2}, 2)$, the path and auxiliary vertices interchangeable, it's not the case for different b 's. Our construction uses this, but only to preserve the topology properties. When we add k auxiliary vertices, we need to ensure that $kb > k - 1$ and $k - 1 > b$. For small b , we change the signal that it does not span two consecutive slices, but one (as on Figure 10).



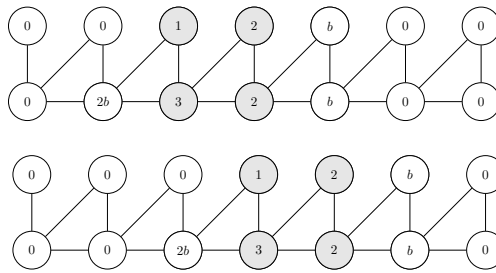
■ **Figure 10** Schematic wire construction for general b with the slices of size k . The numbers are payoffs of important vertices.

B More gadgets

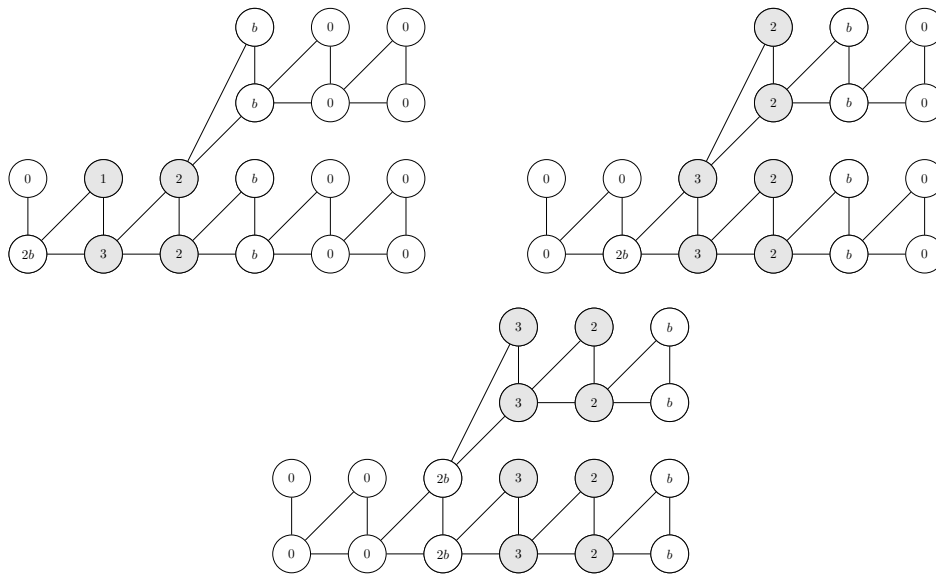
In this section, we provide more detailed figures for some gadgets. Moreover, you can see videos of some gadgets in action at https://pub.ist.ac.at/~jsvoboda/research/spatial_games/.

On Figure 11 and Figure 12 we see explicit graph shown on Figure 3 and Figure 4.

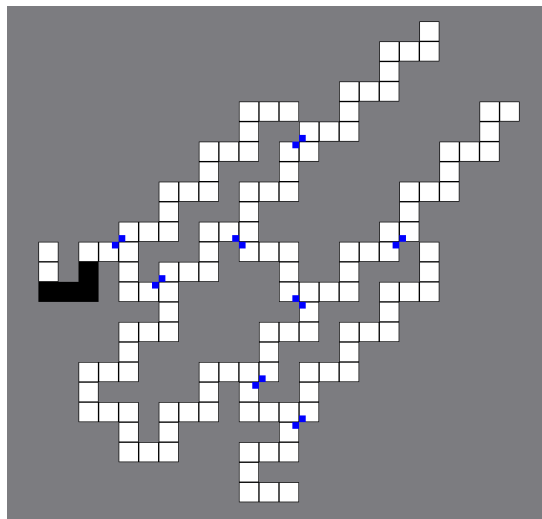
11:14 Complexity of Spatial Games



■ **Figure 11** Sending signal through the wire with explicit payoffs.



■ **Figure 12** Splitting the signal in two. Note that the splitter does not have a direction (a signal from any input/output is split and sent by the other two inputs/outputs).



■ **Figure 13** Crossing of two wires. Signal coming from left leaves the gadget by the right wire and signal coming from bottom leaves the gadget by the top wire.

Robustly Separating the Arithmetic Monotone Hierarchy via Graph Inner-Product

Arkadev Chattopadhyay ✉ 🏠

TIFR, Mumbai, India

Utsab Ghosal ✉

Chennai Mathematical Institute, India

Partha Mukhopadhyay ✉ 🏠

Chennai Mathematical Institute, India

Abstract

We establish an ϵ -sensitive hierarchy separation for monotone arithmetic computations. The notion of ϵ -sensitive monotone lower bounds was recently introduced by Hrubeš [12]. We show the following:

- There exists a monotone polynomial over n variables in VNP that cannot be computed by $2^{o(n)}$ size monotone circuits in an ϵ -sensitive way as long as $\epsilon \geq 2^{-\Omega(n)}$.
- There exists a polynomial over n variables that can be computed by polynomial size monotone circuits but cannot be computed by any monotone arithmetic branching program (ABP) of $n^{o(\log n)}$ size, even in an ϵ -sensitive fashion as long as $\epsilon \geq n^{-\Omega(\log n)}$.
- There exists a polynomial over n variables that can be computed by polynomial size monotone ABPs but cannot be computed in $n^{o(\log n)}$ size by monotone formulas even in an ϵ -sensitive way, when $\epsilon \geq n^{-\Omega(\log n)}$.
- There exists a polynomial over n variables that can be computed by width-4 polynomial size monotone arithmetic branching programs (ABPs) but cannot be computed in $2^{o(n^{1/d})}$ size by monotone, unbounded fan-in formulas of product depth d even in an ϵ -sensitive way, when $\epsilon \geq 2^{-\Omega(n^{1/d})}$. This yields an ϵ -sensitive separation of constant-depth monotone formulas and constant-width monotone ABPs.

The novel feature of our separations is that in each case the polynomial exhibited is obtained from a *graph inner-product* polynomial by choosing an appropriate graph topology. The closely related graph inner-product Boolean function for expander graphs was invented by Hayes [8], also independently by Pitassi [16], in the context of *best-partition* multiparty communication complexity.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebraic Complexity, Discrepancy, Lower Bounds, Monotone Computations

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.12

Funding *Arkadev Chattopadhyay*: Partially supported by a MATRICS grant MTR/2019/001633 of the Science and Engineering Research Board, DST, India.

Utsab Ghosal: Partially supported by Infosys Foundation.

Partha Mukhopadhyay: Partially supported by Infosys Foundation.

Acknowledgements We thank the anonymous reviewers for their feedback.

1 Introduction

While considerable progress has been made in monotone complexity, several fundamental problems remain open. In particular, it is known that monotone lower bounds of a certain kind are enough to imply the major breakthrough of obtaining strong general circuit lower bounds. In Boolean complexity, it has long been known [20] that monotone circuit lower bounds for slice functions are sufficient to yield general lower bounds. In the context of arithmetic complexity, Hrubeš [12] recently formulated an analogous result by showing that ϵ -sensitive monotone lower bounds for arbitrarily small but non-zero ϵ yield lower bounds



© Arkadev Chattopadhyay, Utsab Ghosal, and Partha Mukhopadhyay;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 12; pp. 12:1–12:20



Leibniz International Proceedings in Informatics

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

even for non-monotone circuits. More generally, consider F_n to be the full polynomial, $(1 + x_1 + x_2 + \dots + x_n)^n$, of degree n that obviously has a simple monotone circuit. For any monotone polynomial f , Hrubeš showed that super-polynomial lower bounds on the monotone circuit (branching program, formula) size for $F_n + \epsilon \cdot f$ for arbitrarily small $\epsilon > 0$, yields general lower bounds on circuit (branching program, formula¹) size for computing f . Interestingly, most of the candidate polynomials used in the lower bound literature are set-multilinear. However for general set-multilinear circuits the current best known lower bound is nearly quadratic [1] which is obtained via the lower bound for syntactic multilinear circuits. Before aiming for a general circuit lower bound result via ϵ -sensitive approach, a realistic goal could be to prove strong lower bounds for set-multilinear circuits. It can be observed easily from the result in [12] that setting $F_{k,n} := \prod_{i=1}^k (x_{i,1} + \dots + x_{i,n})$ to be the full set-multilinear polynomial (over $k \times n$ variables) and proving ϵ -sensitive bounds (for sufficiently small ϵ) yields general set-multilinear circuit lower bounds. More concretely, the following theorem is implicit in [12].

► **Theorem 1.1** ([12, Re-statement of Theorem 1 from the work by Hrubeš]). *Let f be a set-multilinear polynomial of degree k which is defined over a matrix of variables $X_{k \times n}$. If f has a set-multilinear circuit of size s then there exists $\epsilon_0 > 0$ such that for every $0 < \epsilon < \epsilon_0$ the polynomial $\prod_{i=1}^k \left(\sum_{j=1}^n x_{i,j} \right) + \epsilon \cdot f$ has a monotone circuit of size $\text{poly}(s, n, k)$.*

In fact in the above theorem $\epsilon_0 \approx \frac{1}{2^{2^s}}$ suffices. Hrubeš argues that proving ϵ -sensitive lower bounds even for moderately small ϵ seems to be non-trivial as it'd require exploiting information of the values of coefficients of monomials appearing in f . Most techniques employed for proving monotone lower bounds ignore the specific values of coefficients. They use the structure of the support set of the monomials alone. With respect to such techniques, the determinant and permanent polynomials, two polynomials that Valiant's VP vs. VNP conjecture asserts have very different complexities, remain equivalent. Some recent works that are able to exploit the values of coefficients are that of Yehudayoff [21] and the work of Srinivasan [18] that builds upon the former. However, these techniques have not yielded so far ϵ -sensitive lower bounds. Such bounds were recently obtained by Chattopadhyay, Datta and Mukhopadhyay [5] and by Chattopadhyay et.al. [4], adapting techniques from 2-party communication complexity.

Motivated towards gaining a better understanding of ϵ -sensitive computations, we revisit the question of separating the powers of some of the key monotone arithmetic models: circuits, branching programs, formulas and constant-depth (unbounded fan-in) formulas. Arvind, Joglekar and Srinivasan [2] proved that constant-depth monotone formulas are strictly less powerful than monotone formulas of unrestricted depth by considering a specialized polynomial. Hrubeš and Yehudayoff [10] showed that elementary symmetric polynomials cannot be computed in polynomial size by monotone formulas. This provides a separation of the powers of monotone formulas from that of monotone ABPs. Later, a different work of Hrubeš and Yehudayoff [11] showed a similar separation of the power of monotone ABPs and monotone circuits. This required the construction of an altogether different polynomial. Very recently, Komarath, Pandey and Rahul [13] provided a unified treatment of these separations

¹ The case of formulas comes with the following subtlety: Hrubeš' argument uses a homogenization trick. Unlike circuits or ABPs, we don't know yet if formulas can be homogenized without significant blow-up [17, Subsection 5.1]. This seemingly prevents a direct application of Hrubeš' argument to formulas. Nevertheless, using the fact that size s ABPs can be simulated by size $s^{\log s}$ formulas, one concludes quite easily that an ϵ -sensitive monotone lower bound for formulas of the form $n^{(\log n)^{1+\delta}}$ for any $\delta > 0$, is sufficient to imply super-polynomial lower bounds even for general ABPs.

(not including the separation between constant-depth formulas and formulas of unrestricted depth) by making use of graph homomorphism polynomials. Hence, it is natural to ask if these separations can be strengthened or made robust in the following way: can we exhibit a polynomial f that has polynomial size monotone circuits (ABPs) but even $F_n + \epsilon \cdot f$ has no polynomial size monotone ABP (formula)?

The main contribution of this work is to provide such robust separations between the power of monotone circuits, ABPs, formulas and constant-depth circuits in a unified way. The full polynomial is used for the set-multilinear setting and we are able to handle fairly low range for the parameter ϵ in our results. Our separations build on the connection developed in [5, 4] between randomized communication complexity and ϵ -sensitive lower bounds. In particular, this allows us to exhibit a general framework to define *graph inner-product* polynomials such that simply changing the graph appropriately yields the required polynomial for each of our separations. More precisely, given an undirected graph G on k vertices and a number m , we first define a Boolean function, called the graph inner-product function and denoted by $\text{IP}_G : \{0, 1\}^{k \times m} \rightarrow \{1, -1\}$. Let $V(G) := \{u_1, \dots, u_k\}$. We identify each vertex with a variable \vec{u}_i that takes m -bit binary vectors as values. Then,

$$\text{IP}_G(\vec{u}_1, \dots, \vec{u}_k) := \left(-1 \right)^{\sum_{(u_i, u_j) \in E(G)} \langle \vec{u}_i, \vec{u}_j \rangle}$$

where $\forall (u_i, u_j) \in E(G)$, $\langle \vec{u}_i, \vec{u}_j \rangle := \sum_{t \in [m]} u_t^i \cdot u_t^j$ and $\vec{u}_i = (u_1^i, u_2^i, \dots, u_m^i) \in \{0, 1\}^m$.

We consider a set-multilinear polynomial over an input matrix X of dimension $k \times n$ with entries $X[i, j] := x_{i,j}$ of indeterminates, and $n = 2^m$. The monomials will naturally encode satisfying assignments to the Boolean graph inner product function defined above. Towards this, define $\mathbb{M}[X]$ to be the set of all set-multilinear monomials of degree k over $X = \{X_i \mid i \in [k]\}$, where $\forall i X_i = \{x_{i,j} \mid j \in [n]\}$. With every map $\nu : [k] \rightarrow [n]$, we identify a monomial $\kappa_\nu \in \mathbb{M}[X]$ as $m_\nu := \prod_{i=1}^k x_{i, \nu(i)}$. This forms a bijection between set $\mathbb{M}[X]$ and $\mathbb{T} = \{\nu \mid \nu : [k] \rightarrow [n]\}$. Now each map $\nu \in \mathbb{T}$ can be identified by a k tuple of m -bit vectors $(\vec{\nu}_1, \dots, \vec{\nu}_k)$ where for every $i \in [k]$ $\vec{\nu}_i$ is the binary representation of $\nu(i) \in [n]$. So in this way, given map $\nu : [k] \rightarrow [n]$, any set-multilinear degree k monomial $\kappa = x_{1, \nu(1)} \cdots x_{k, \nu(k)}$ corresponds to a k tuple of m -bit vectors $\tilde{\kappa} = \{\vec{\nu}_1, \dots, \vec{\nu}_k\}$ where each $\vec{\nu}_i$ is the binary representation of $\nu(i)$.

Let,

$$f_{G,m} := \sum_{\text{IP}_G(\tilde{\kappa}) = -1} \kappa$$

be called the $\text{IP}_{G,m}$ polynomial.

Further, let

$$F_{k,n} := \prod_{i=1}^k (x_{i,1} + \cdots + x_{i,n}),$$

be the full set-multilinear polynomial over $\mathbb{M}[X]$.

We can now state our first theorem that implies the first strongly exponential ϵ -sensitive monotone lower bounds for an explicit (monotone) polynomial in VNP. Recall that $n = 2^m$.

► **Theorem 1.2.** *Let G be a constant-degree expander graph on k vertices. Then, there exists a constant $c > 0$ such that any monotone circuit computing either of the polynomial $F_{k,n} \pm \epsilon \cdot f_{G,m}$ has size $2^{\Omega(km)}$ as long as $\epsilon \geq 2^{-ckm}$.*

► **Remark 1.3.** Plugging $m = 1$ in Theorem 1.2, we recover the claimed strongly exponential lower bound as long as $\epsilon = 2^{-\Omega(n)}$.

The VNP upper bound for the polynomial $f_{G,m}$ follows from Valiant’s criterion [19, Proposition 4].

Our second theorem obtains an ϵ -sensitive separation between monotone circuits and monotone ABPs.

► **Theorem 1.4.** *Let T be the full binary tree on k vertices. Then, $f_{T,m}$ can be computed by monotone circuits of size $O(kn^3)$. On the other hand, there exists a constant $c > 0$ such that any monotone ABP computing either of the polynomial $F_{k,n} \pm \epsilon \cdot f_{T,m}$ has size $k^{\Omega(m)}$ as long as $\epsilon \geq k^{-cm}$.*

We next provide an analogous separation between monotone ABPs and monotone formulas by considering a path.

► **Theorem 1.5.** *Let Γ be a simple path on k vertices. Then, the polynomial $f_{\Gamma,m}$ can be computed by a monotone ABP of size $O(kn)$. On the other hand, there exists a constant $c > 0$ such that any monotone formula computing either of the polynomial $F_{k,n} \pm \epsilon \cdot f_{\Gamma,m}$ has size $k^{\Omega(m)}$ as long as $\epsilon \geq k^{-cm}$.*

► **Remark 1.6.** Dvir et al. [7] exhibited a monotone multilinear polynomial that is computable by a polynomial size monotone ABP but requires $n^{\Omega(\log n)}$ size to be computed by every multilinear formula. As their lower bound on multilinear formula size is obtained via rank based arguments, it also implies ϵ -sensitive monotone lower bounds of $n^{\Omega(\log n)}$, even for arbitrarily small non-zero ϵ . However, let us recall that the polynomial used by Dvir et al. is a more involved polynomial that is not based on the inner-product function.

Finally, we provide a separation between constant-depth monotone formulas and monotone formulas. In fact, we provide a stronger separation, that of monotone constant-depth formulas and constant-width ABPs.

► **Theorem 1.7.** *Let Γ be a simple path on k vertices. Then, the polynomial $f_{\Gamma,1}$ can be computed by a monotone width-4 ABP of size $O(k)$. On the other hand, there exists a constant $c > 0$ such that any monotone formula of product-depth d computing either of the polynomial $F_{k,2} \pm \epsilon \cdot f_{\Gamma,1}$ has size $2^{\Omega(k^{1/d})}$ as long as $\epsilon \geq 2^{-ck^{1/d}}$.*

► **Remark 1.8.** It is known that poly-size bounded-depth monotone formulas can be simulated by poly-size bounded-width monotone ABPs which in turn can be simulated by monotone formulas of unrestricted depth and polynomial size. Our result, thus in particular, shows that the class of polynomials computed by constant-depth monotone formulas of polynomial size are strictly contained (in a strong sense) in the class of polynomials having bounded-width monotone ABPs of polynomial size. The work of Nisan and Wigderson [15] already implies such a separation, even for arbitrarily small ϵ , but uses a different polynomial. More precisely, they show that any product-depth d set-multilinear circuit for computing the iterated matrix multiplication polynomial (over 2×2 matrices) $\text{IMM}_{2,n}$ must be of size $2^{\Omega(n^{1/d})}$.

1.1 Outline of our Technique

We build upon the insight relating ϵ -sensitive lower bounds with the measure of discrepancy under universal distributions, originating in [5, 4]. Both of these works prove lower bounds against monotone circuits for polynomials that are not known to have any efficient monotone computation. The main concern here is to prove separations in the monotone hierarchy.

Thus, the new challenge is to come up with far “*easier polynomials*” to which a discrepancy like measure can still be applied. The canonical example of a “function” to which the discrepancy method applies in communication complexity is that of Inner-product. However, one important difference between the setting of standard 2-party communication complexity and arithmetic circuits is that while in the former, every rectangle that appears in a rectangular decomposition conforms to the same partition of inputs among the players, in the latter each product polynomial appearing in a decomposition is free to have its own partition of the input variables. Indeed, the standard Inner-product function becomes trivial for the best (w.r.t the players) partition.

This is why we turn to best-partition communication complexity, a slightly non-standard model. The relevance of considering graph inner-product based functions was also pointed out by Hrubeš and Yehudayoff [11]. Hayes [8], and independently Pitassi [16], designed an appropriate version of the Inner-product function that they called Graph inner-product (see the definition from the Introduction) which they proved is hard against all (balanced) partitions. To do this, they chose the graph to be a constant-degree expander. A standard property of such an expander is that the size of any balanced cut is large. This allows one to say that for every possible partition of inputs among the players, one can induce a large copy of the standard inner-product as a sub-function for which the given partition is the worst for the players. This does not immediately give us a monotone arithmetic circuit lower bound. To get there, we need to argue against *many partitions appearing together* in the decomposition. This is where we use the idea from [4] of discrepancy w.r.t universal distributions that is a measure which is partition independent. This gives us our Theorem 1.2, an ϵ -sensitive strongly exponential lower bounds against monotone circuits.

How does one tune the complexity of a graph Inner-product polynomial so that it becomes easy for monotone circuits but remains robustly hard against monotone branching programs? The starting point is to look at the (not robust) separation of these two classes by Hrubeš and Yehudayoff [11]. They showed that there is a subtle difference between decomposition theorems for ABPs and that of circuits. ABPs give rise to a slightly more structured decomposition where, for every r , we can force each product polynomial to have a partition such that one part has size exactly r . They further showed that for the full binary tree on k vertices, any cut where one side has $r(k)$ vertices, has size $\Omega(k)$. Exploiting this insight, we establish Theorem 1.4 by proving a universal discrepancy bound for all such partitions on the binary-tree inner-product function. It is to be noted that [11] also describes a general connection between the choice of a graph and the properties of the corresponding inner-product function.

To separate formulas from ABPs, we use the fact that the decomposition theorem for formulas provides even more structure. Here, every polynomial of degree k appearing in a decomposition can be written as a product of about $\log k$ -many polynomials rather than two. This corresponds, with the usual caveat of mixed vs. best partition, to the case of the best-partition $\log k$ -party number-in-the-hand model of randomized communication complexity. While the goal of having an efficient ABP upper bound for the polynomial forces the corresponding 2-party game to be easy for at least some partition, choosing the graph to be a simple path ends up having the following simple but remarkable feature: for every balanced $\log k$ -wise partition of the inputs, one can design a 2-party game with a hard partition. This reduces the task to proving a discrepancy bound for this hard 2-party partition which drives Theorem 1.5.

To separate monotone ABPs of constant width from monotone constant-depth (but unbounded fan-in) formulas, we first note that keeping the path inner-product polynomial becomes easy for ABPs of constant width once we restrict each node in the path to have

two (or any constantly many) variables instead of n . Finally, we exploit the super-structured decomposition theorem for constant-depth (but unbounded fan-in) formulas. Each product polynomial now is the product of $k^{1/d}$ -many set-multilinear product polynomials, where d is the product-depth of the formula. Using similar ideas as in the proof of Theorem 1.5 with this additional structure, we establish appropriate discrepancy bounds to yield Theorem 1.7.

Organization

The paper is organized as follows. We provide some background mainly on communication complexity and monotone algebraic complexity in Section 2. In Section 3, we prove results on discrepancy bounds for graph inner product function. The proof of Theorem 1.2 is given in Section 4 that shows ϵ -sensitive lower bound for monotone circuits. Section 5 contains the proof of Theorem 1.4 which shows the separation between monotone circuits and ABPs. Finally in Section A, we provide the proofs of Theorem 1.5 and Theorem 1.7 establishing the separations between monotone ABPs vs formulas, and constant width ABPs vs constant depth formulas.

2 Preliminaries

Notation

Let $[n] = \{1, 2, \dots, n\}$. Polynomials are always considered over $\mathbb{R}[X]$ where \mathbb{R} is the set of reals.

Set-Multilinear Polynomials

Let $X = \bigsqcup_{i=1}^k X_i$ be a set of variables where $X_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,n}\}$. A polynomial $f \in \mathbb{R}[X]$ is set-multilinear if each monomial in f respects the partition given by the set of variables X_1, X_2, \dots, X_k . In other words, each monomial κ in f is of the form $x_{1,j_1} x_{2,j_2} \cdots x_{k,j_k}$. For the purpose of the paper it is also useful to think the variables are from a matrix $M_{k \times n}$ where the i^{th} row is $\{x_{i,1}, x_{i,2}, \dots, x_{i,n}\}$.

Ordered Polynomials

For a monomial of the form $m = x_{i_1,j_1} x_{i_2,j_2} \cdots x_{i_k,j_k}$ we define the set $I(m) = \{i_1, i_2, \dots, i_k\}$. If a polynomial f has the same set $I(m)$ for every monomial occurring in it with a nonzero coefficient, then we say that the polynomial is ordered and we write $I(f) = I(m)$ for each m . Clearly, the set-multilinear polynomials are ordered polynomials with $I(f) = \{1, 2, \dots, k\}$.

Set-Multilinear Circuits

We recall the definition of set-multilinear circuits [3]. The definition can be extended to set-multilinear ABPs and formulas naturally.

► **Definition 2.1.** *A circuit computing a set-multilinear polynomial f over the variable set $X = \bigsqcup_{i=1}^n X_i$ is called set-multilinear if at every gate the circuit computes a set-multilinear polynomial with respect to an induced partition.*

More precisely, for every node u in the circuit we can associate a set of indices $I_u \subseteq [k]$ such that the set-multilinear polynomial f_u computed at node u is defined over $\bigsqcup_{i \in I_u} X_i$. So for any node $I_u = I(f_u)$. If u is a $+$ node with children u_1, u_2 then $I_u = I_{u_1} = I_{u_2} = I_u$. If u is \times node with children u_1, u_2 then $I_u = I_{u_1} \sqcup I_{u_2}$.

2.1 Structure of Monotone Circuits

The main structural result for monotone circuits that we use throughout, is the following theorem.

► **Theorem 2.2** ([21, Lemma 1]). *Let $n > 2$ and $f \in \mathbb{R}[X]$ be an ordered monotone polynomial with $I(p) = [k]$. Let C be a monotone circuit of size s that computes f . Then, we can write $f = \sum_{t=1}^s a_t \cdot b_t$ where a_t and b_t are monotone ordered polynomials with $\frac{k}{3} \leq |I(a_t)| \leq \frac{2k}{3}$ and $I(b_t) = [k] \setminus I(a_t)$. Moreover, $a_t b_t \leq f$ for each $1 \leq t \leq s$, by which we mean that the coefficient of any monomial in $a_t b_t$ is bounded by the coefficient of the same monomial in f .*

A partition $P = (A, B)$ of $[k]$ is said to be perfectly balanced if $|A| = |B| = \frac{k}{2}$ and is said to be nearly balanced if $\frac{k}{3} \leq |A|, |B| \leq \frac{2k}{3}$. An ordered product polynomial $a \cdot b$ on n variables is said to be nearly balanced if $\frac{k}{3} \leq |I(a)|, |I(b)| \leq \frac{2k}{3}$.

2.2 Structure of Monotone ABPs

We recall the definition of algebraic branching programs (ABPs).

► **Definition 2.3** (Algebraic Branching Program). *An algebraic branching program (ABP) is a layered directed acyclic graph. The vertex set is partitioned into layers $0, 1, \dots, k$, with directed edges only between adjacent layers (i to $i + 1$). There is a source vertex of in-degree 0 in layer 0, and one out-degree-0 sink vertex in layer k . Each edge is labeled by an affine \mathbb{F} -linear form where \mathbb{F} is the underlying field. The polynomial computed by the ABP is the sum over all source-to-sink directed paths of the ordered product of affine forms labeling the path edges.*

The following structure theorem is well-known.

► **Theorem 2.4** ([11, Lemma 3]). *Let f be a degree k monotone set-multilinear polynomial computed by a size s ABP. Then for every $j \in [k]$ there exists s pairs of monotone set-multilinear polynomials $\{g_i, h_i \mid i \in [s]\}$ such that $f = \sum_{i=1}^s g_i \cdot h_i$ where for every i , $|I(g_i)| = j$ and $|I(h_i)| = k - j$. $(I(g_i), I(h_i))$ gives a partition of $[k]$.*

2.3 Structure of (Monotone) Set-Multilinear Formulas

► **Definition 2.5** ((Monotone) Set-Multilinear-log-Product Polynomials.). *A degree k polynomial f defined over a $k \times n$ matrix M of variables is called a (monotone) set-multilinear-log-product polynomial if there exists p (monotone) set-multilinear polynomials f_1, \dots, f_p such that the following holds.*

1. $f = \prod_{i=1}^p f_i$.
2. $\forall i \in [p - 1], \left(\frac{1}{3}\right)^i k \leq |I(f_i)| \leq \left(\frac{2}{3}\right)^i k$ where $I(f_i)$ is the set of rows of M on which the polynomial f_i is defined.
3. $\forall i \neq j, I(f_i) \cap I(f_j) = \emptyset$.
4. $|I(f_p)| = 1$.

The following structure theorem is well-known and proved in [10]. However, we include a self-contained proof in the appendix for completeness.

► **Theorem 2.6** ([10, Lemma 4]). *Let f be a degree k set-multilinear polynomial computed by a (monotone) formula of size s . Then there exists (monotone) set-multilinear-log-product polynomials $g_1, g_2, \dots, g_{s'}$ such that $s' \leq s$, $f = g_1 + g_2 + \dots + g_{s'}$.*

2.4 Structure of (Monotone) Set-Multilinear Constant Depth Formula

► **Definition 2.7** ((p, ℓ) -Form). *A degree k (monotone) set-multilinear polynomial f defined over a matrix $M_{k \times n}$ of variables has a (p, ℓ) -form if there exists p (monotone) set-multilinear polynomials f_1, \dots, f_p such that the following holds.*

1. $f = \prod_{i=1}^p f_i$.
2. $\forall i \in [p], |I(f_i)| \geq \ell$, where $I(f_i)$ is the set of rows of $M_{k \times n}$ on which the polynomial f_i is defined.
3. $\forall i \neq j, I(f_i) \cap I(f_j) = \emptyset$.

The following theorem is a re-statement of Lemma 9 in [10]. For completeness, the proof is included in the appendix.

► **Theorem 2.8** ([10, Lemma 9]). *Let f be a degree k set-multilinear polynomial computed by a (monotone) formula of size s and product depth d . Let $q > 1$ be a natural number such that $k > (2q)^d$. Then there exists (monotone) set-multilinear- $(q, k(2q)^{-d})$ -form polynomials $g_1, g_2, \dots, g_{s'}$ such that $s' \leq s$, $f = g_1 + g_2 + \dots + g_{s'}$.*

2.5 Communication Complexity

We recall some basic results from communication complexity. The details can be found in [14]. Let us very briefly first recall basic notions in the 2-party communication model of Yao. The joint input space of Alice and Bob is $\{0, 1\}^m \times \{0, 1\}^m$ with each player receiving an m -bit Boolean string, and they want to evaluate a Boolean function $F : \{0, 1\}^m \times \{0, 1\}^m \rightarrow \{-1, 1\}$. One defines a combinatorial rectangle R as a product set $A \times B$, for some $A, B \subseteq \{0, 1\}^m$. Put another way, R is just a sub-matrix of the $2^m \times 2^m$ communication matrix M_F of the function F , that Alice and Bob want to compute. The rows of this matrix are indexed by possible inputs of Alice and the columns by the ones of Bob and $M_F(x, y) = F(x, y)$. One of the important notions is discrepancy. For a rectangle R , the discrepancy $\text{Disc}_\delta(F, R) := |\delta(R \cap F^{-1}(1)) - \delta(R \cap F^{-1}(-1))|$ where δ is a distribution on the input space $\{0, 1\}^m \times \{0, 1\}^m$. In other words,

$$\text{Disc}_\delta(F, R) = \left| \mathbb{E}_{(x,y) \sim \delta} [F(x, y)R(x, y)] \right|.$$

The discrepancy of F under δ is defined as

$$\text{Disc}_\delta(F) := \max_R \text{Disc}_\delta(F, R).$$

In this work, we will be forced to look at variable partition models. That is, the n input bits will be partitioned among Alice and Bob in multiple ways. Each such partition P has its own set of rectangles, denoted by $\mathcal{R}(P)$. Hence, we define,

$$\text{Disc}_{\delta, P}(F) := \max_{R \in \mathcal{R}(P)} \text{Disc}_\delta(F, R).$$

The 2-party model extends to t -party model naturally. Here, we have t players P_1, P_2, \dots, P_t and the joint input space is $\{0, 1\}^{m_1} \times \{0, 1\}^{m_2} \times \dots \times \{0, 1\}^{m_t}$. Player P_i receives an input from $\{0, 1\}^{m_i}$. Together, they want to compute a function $F : \{0, 1\}^{m_1} \times \{0, 1\}^{m_2} \times \dots \times \{0, 1\}^{m_t} \rightarrow \{-1, 1\}$. We can similarly define a combinatorial rectangle $R = R_1 \times \dots \times R_t$ where each $R_i \subseteq \{0, 1\}^{m_i}$. For any distribution δ over the input space $\{0, 1\}^{m_1} \times \{0, 1\}^{m_2} \times \dots \times \{0, 1\}^{m_t}$ and a rectangle R we define

$$\text{Disc}_\delta^t(F, R) := |\delta(R \cap F^{-1}(1)) - \delta(R \cap F^{-1}(-1))|. \quad (1)$$

Now we define the discrepancy of F in the following way,

$$\text{Disc}_\delta^t(F) := \max_R \text{Disc}_\delta^t(F, R)$$

Exactly like in the two-party case, we will be considering the t -party discrepancy w.r.t multiple t -way partitions. Hence, given such a partition P , we analogously define $\text{Disc}_{\delta, P}^t(F)$. Let us recall the inner product function,

$$\text{IP}_m(x, y) := \left(-1 \right)^{\sum_{i=1}^m x_i y_i},$$

that is widely studied. It is well-known that the two-party discrepancy of the inner product function is small under the uniform distribution \mathcal{U} over $\{0, 1\}^m \times \{0, 1\}^m$. This was first proved by Chor and Goldreich [6]. A self-contained proof can be found in [14].

► **Theorem 2.9** ([14, Example 3.29]). *Under the uniform distribution \mathcal{U} over $\{0, 1\}^m \times \{0, 1\}^m$, $\text{Disc}_{\mathcal{U}}(\text{IP}_m) = 2^{-\Omega(m)}$.*

3 Discrepancy Bound for Graph Inner Product

3.1 Graph Inner Product Function

Given a graph G on k vertices $\{u_1, u_2, \dots, u_k\}$, we identify each vertex u_i with variable \vec{u}_i which takes m -bit binary vectors as values. Now we define the following function

$$\text{IP}_G(\vec{u}_1, \dots, \vec{u}_k) = \left(-1 \right)^{\sum_{(u_i, u_j) \in E(G)} \langle \vec{u}_i, \vec{u}_j \rangle}$$

where $\forall (u_i, u_j) \in E(G)$ and $\langle \vec{u}_i, \vec{u}_j \rangle = \sum_{t \in [m]} u_t^i \cdot u_t^j$ and $\vec{u}_i = (u_1^i, u_2^i, u_3^i, \dots, u_m^i) \in \{0, 1\}^m$.

3.2 Graph Inner Product Polynomial

Consider the input matrix X of dimension $k \times n$ with entries $X[i, j] := x_{i,j}$ of indeterminates, and $n = 2^m$. Define $\mathbb{M}[X]$ to be the set of all set-multilinear monomials of degree k over $X = \{X_i \mid i \in [k]\}$, where $\forall i X_i = \{x_{i,j} \mid j \in [n]\}$. With every map $\nu : [k] \rightarrow [n]$, we identify a monomial $\kappa_\nu \in \mathbb{M}[X]$ as $\kappa_\nu := \prod_{i=1}^k x_{i, \nu(i)}$. This forms a bijection between set $\mathbb{M}[X]$ and $\mathbb{T} = \{\nu \mid \nu : [k] \rightarrow [n]\}$.

Now each map $\nu \in \mathbb{T}$ can be identified by a k tuple of m -bit vectors $(\vec{\nu}_1, \dots, \vec{\nu}_k)$ where for every $i \in [k]$ $\vec{\nu}_i$ is the binary representation of $\nu(i) \in [n]$. So in this way, given map $\nu : [k] \rightarrow [n]$, any set-multilinear degree k monomial $\kappa = x_{1, \nu(1)} \cdots x_{k, \nu(k)}$ corresponds to a k tuple of m -bit vectors $\vec{\kappa} = \{\vec{\nu}_1, \dots, \vec{\nu}_k\}$.

Let the polynomial,

$$f_{G,m} := \sum_{\text{IP}_G(\vec{\kappa}) = -1} \kappa$$

be denoted as $\text{IP}_{G,m}$ and be called the G -Inner product polynomial.

3.3 Discrepancy and Lower Bound Correspondence

Recently in [4], a correspondence between ϵ -sensitive monotone lower bound for a $0 - 1$ coefficient polynomial f and an appropriate communication problem has been established via the discrepancy measure. There the authors only considered the two-party communication

12:10 Robustly Separating the Arithmetic Monotone Hierarchy via Graph Inner-Product

problem. Here we extend it to t -party communication problem for $t \geq 2$. Let (A_1, A_2, \dots, A_t) be any partition of $[k]$ and let there be t players P_1, \dots, P_t . The players are given a map $\nu : [k] \rightarrow [n]$ in a distributed fashion, i.e., P_i gets a map $\nu_i : A_i \rightarrow [n]$. They jointly want to decide if the monomial κ_ν is present in polynomial f or not. In other words, the players want to compute a Boolean function, denoted by $C^f : \{0, 1\}^{k \times m} \rightarrow \{1, -1\}$, where $C^f(\nu(1), \dots, \nu(k)) = -1$ if monomial κ_ν has coefficient 1 in f and otherwise C^f evaluates to 1. Inspecting the proof in [4], it is easy to observe that the lower bound technique is independent of the monotone computation model. In particular, it applies to ABPs, formulas and constant depth formulas using their respective structure theorems. that is Theorem 2.4, Theorem 2.6 and Theorem 2.8. More precisely, we can restate their result in the following form.

► **Theorem 3.1** ([4, Adaptation of their Theorem 1.3]). *Let f be any 0 – 1 set-multilinear monotone polynomial defined over a matrix of variables of dimension $k \times n$. Let Δ be a distribution over $[k]^n$.*

1. *The monotone circuit complexity of $F_{k,n} - \epsilon \cdot f$ (resp. $F_{k,n} + \epsilon \cdot f$) is at least $\frac{\epsilon}{3\gamma}$ (resp. $\frac{\epsilon}{6\gamma}$) as long as $\epsilon \geq \frac{6\gamma}{1-3\gamma}$ (resp. $\epsilon \geq \frac{6\gamma}{1-12\gamma}$), where $\gamma := \max_P \text{Disc}_{\Delta,P}(C^f)$ and P is any nearly balanced 2-wise partition of $[k]$.*
2. *Let $r \in [k]$. Then the monotone ABP complexity of $F_{k,n} - \epsilon \cdot f$ (resp. $F_{k,n} + \epsilon \cdot f$) is at least $\frac{\epsilon}{3\gamma}$ (resp. $\frac{\epsilon}{6\gamma}$) as long as $\epsilon \geq \frac{6\gamma}{1-3\gamma}$ (resp. $\epsilon \geq \frac{6\gamma}{1-12\gamma}$), where $\gamma := \max_P \text{Disc}_{\Delta,P}(C^f)$. Here the max runs over all 2-wise partitions $P = (A, B)$ of $[k]$ such that $|A| = r$.*
3. *The monotone formula complexity of $F_{k,n} - \epsilon \cdot f$ (resp. $F_{k,n} + \epsilon \cdot f$) is at least $\frac{\epsilon}{3\gamma}$ (resp. $\frac{\epsilon}{6\gamma}$) as long as $\epsilon \geq \frac{6\gamma}{1-3\gamma}$ (resp. $\epsilon \geq \frac{6\gamma}{1-12\gamma}$), where $\gamma := \max_P \text{Disc}_{\Delta,P}^t(C^f)$, P runs over all t -wise partitions with $t = \Omega(\log k)$.*
4. *The monotone product depth d formula complexity of $F_{k,n} - \epsilon \cdot f$ (resp. $F_{k,n} + \epsilon \cdot f$) is at least $\frac{\epsilon}{3\gamma}$ (resp. $\frac{\epsilon}{6\gamma}$) as long as $\epsilon \geq \frac{6\gamma}{1-3\gamma}$ (resp. $\epsilon \geq \frac{6\gamma}{1-12\gamma}$), where $\gamma := \max_P \text{Disc}_{\Delta,P}^t(C^f)$, P runs over all t -wise partitions with $t = \Omega(k^{\frac{1}{d}})$.*

3.4 Good Matching in Graphs

Consider a graph G on vertex set V . For a partition $P = (V_1, V_2)$ of V , let G_P be the induced bipartite graph, i.e., $E(G_P) := \{(u, v) : (u, v) \in E(G), u \in V_1, v \in V_2\}$.

► **Definition 3.2.** *Let G be a graph and $P = (V_1, V_2)$ be any partition of the vertex set. A matching M in the induced bipartite graph G_P is called good if for every pair of edges $(u_i, w_i), (u_j, w_j)$ in M , none of $(u_i, u_j), (u_i, w_j), (w_i, w_j)$ and (u_j, w_i) are in $E(G)$. Further, we define*

$$\tau(G, P) = \max_{M \text{ is good w.r.t } P} |M|.$$

► **Lemma 3.3.** *Let G be any graph with maximum degree d . Then for any partition P of the vertex set of G , we have $\tau(G, P) \geq \frac{|E(G_P)|}{2d^2}$.*

Proof. Consider the graph G_P and build a matching $M \subseteq E(G_P)$ by the following process, starting with an empty M .

1. If $E(G_P)$ is empty, return M . Otherwise, add any edge (u, v) in $E(G_P)$ to M .
2. Remove every edge currently in $E(G_P)$ that is incident to a vertex that is a neighbour of u or v (including themselves) in G .
3. Go back to 1.

Observe that after each completion of step 2, the number of edges newly added to M is 1 and the number of edges removed from $E(G_P)$ is at most $2d^2$ since degree of a vertex in G (and therefore in G_P as well) is at most d . Thus, size of M is at least $\frac{|E(G_P)|}{2d^2}$. It's simple to verify that the pruning done at step 2 ensures M forms a good matching. ◀

The following lemma gives a lower bound on the size of a good matching in a constant degree expander graph. The proof follows by a simple application of the Expander Mixing Lemma [9, Lemma 2.5] and Lemma 3.3. Similar arguments also appear in [16, Lemma 4.2] and in [8]. We provide a proof for the sake of completeness in the appendix.

► **Lemma 3.4** ([16, 8]). *Let d be a constant and G be a d regular expander graph on k vertices with the second largest eigen value of the normalized adjacency matrix $\lesssim \frac{1}{\sqrt{d}}$ and $P = (V_1, V_2)$ be a nearly balanced partition of the vertex set V . Then, $\tau(G, P) = \Omega_d(k)$.*

The notation Ω_d hides a constant that depends on d .

The next lemma is about good matching in a full binary tree.

► **Lemma 3.5.** *Given a full binary tree T on k vertices, there exists a number $t \approx \frac{2}{3}k$, such that for any partition $P = (V_1, V_2)$ of the vertex set with $|V_1| = t$, $\tau(T, P) = \Omega(\log k)$.*

Proof. In [11], it is shown that there exists a number $t \approx \frac{2}{3}k$ such that for any partition $P = (V_1, V_2)$ with $|V_1| = t$, the induced bipartite graph T_P has $\Omega(\log k)$ many edges. Since the maximum degree of T is 3, using Lemma 3.3 $\tau(T, P) = \Omega(\log k)$. ◀

3.5 A Communication Problem

Much of what we're going to prove in this section, derives its intuition from the following communication problem in Yao's 2-party model. We state the problem below, although we point out to the reader that if one is just interested in verifying the monotone lower bounds we claim for the various arithmetic models, then it is not necessary to know this communication problem.

► **Problem 3.6.** *Given a graph G with a vertex set V , where V is partitioned into V_1, V_2 and $|V| = k$, we consider the following communication problem in Yao's 2-party model: Alice (Bob) gets the assignment to variables corresponding to vertices of V_1 (V_2). Together they need to evaluate the Boolean function IP_G on their joint inputs.*

Having stated the problem, we prove below the key discrepancy upper bound that shows the above problem's communication complexity, w.r.t. any balanced partition, to be high. This consequence of our discrepancy bound may be of independent interest.

► **Lemma 3.7.** *Consider a graph G on vertex set V such that $V = \{u_1, u_2, \dots, u_k\}$. For a partition $P = (V_1, V_2)$ of V , let G_P be the induced bipartite graph. Under the uniform distribution \mathcal{U} over $\{0, 1\}^{km}$, the following holds: $\text{Disc}_{\mathcal{U}, P}(\text{IP}_G) \leq 2^{-\Omega(\tau(G, P) \cdot m)}$.*

Proof. Let M_P be the good matching in G_P such that $\tau(G, P) = |M_P|$. For convenience let $|M_P| = t$. For any edge (u_i, u_j) in the matching M_P define $\vec{C}_i = \bigoplus_{u_r \in \text{Nbd}(u_i) \setminus \{u_j\}} \vec{u}_r$ and $\vec{C}_j = \bigoplus_{u_\ell \in \text{Nbd}(u_j) \setminus \{u_i\}} \vec{u}_\ell$. Here $\text{Nbd}(u_i) = \{u_\ell : (u_i, u_\ell) \in E(G)\}$. Then, $\text{IP}_G(\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k) = (-1)^D$ where

12:12 Robustly Separating the Arithmetic Monotone Hierarchy via Graph Inner-Product

$$\begin{aligned}
D &= \left(\sum_{(u_i, u_j) \in M_P} (\langle \vec{u}_i, \vec{u}_j \rangle + \langle \vec{C}_i, \vec{u}_i \rangle + \langle \vec{C}_j, \vec{u}_j \rangle) \right) + \sum_{\substack{u_q, u_r \notin V(M_P) \\ (u_q, u_r) \in E(G)}} \langle \vec{u}_q, \vec{u}_r \rangle \\
&= \left(\sum_{(u_i, u_j) \in M_P} (\langle \vec{u}_i + \vec{C}_j, \vec{u}_j + \vec{C}_i \rangle + \langle \vec{C}_i, \vec{C}_j \rangle) \right) + \sum_{\substack{u_q, u_r \notin V(M_P) \\ (u_q, u_r) \in E(G)}} \langle \vec{u}_q, \vec{u}_r \rangle \\
&= \left(\sum_{(u_i, u_j) \in M_P} \langle \vec{u}_i, \vec{u}_j \rangle + c \right) \tag{2}
\end{aligned}$$

Here $\vec{u}_i' = \vec{u}_i + \vec{C}_j$ and $\vec{u}_j' = \vec{u}_j + \vec{C}_i$. Note that

$$c = \sum_{\substack{u_q, u_r \notin V(M_P) \\ (u_q, u_r) \in E(G)}} \langle \vec{u}_q, \vec{u}_r \rangle + \sum_{(u_i, u_j) \in M_P} \langle \vec{C}_i, \vec{C}_j \rangle.$$

For the partition $P = (V_1, V_2)$ consider an arbitrary rectangle $R \in \mathcal{R}(P)$ in $\{0, 1\}^{|V_1| \cdot m} \times \{0, 1\}^{|V_2| \cdot m}$. Here for the sake of simplicity we abuse the notation R and denote it as a characteristic function for the rectangle R . Let \mathcal{U} be the uniform distribution over $\{0, 1\}^{|V_1| \cdot m} \times \{0, 1\}^{|V_2| \cdot m}$ and \mathcal{U}_m be the uniform distribution over $\{0, 1\}^m$.

Notice that

$$\begin{aligned}
\text{Disc}_{\mathcal{U}, P}(\text{IP}_G, R) &= \left| \mathbb{E}_{\vec{u}_i \sim \mathcal{U}_m} \left[\text{IP}_G(\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k) \cdot R(\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k) \right] \right| \\
&= \left| \mathbb{E}_{\vec{u}_i \sim \mathcal{U}_m: u_i \notin M_P} \left[\mathbb{E}_{\vec{u}_i \sim \mathcal{U}_m: u_i \in M_P} \left[\text{IP}_G(\vec{u}_1, \dots, \vec{u}_k) R(\vec{u}_1, \dots, \vec{u}_k) \right] \right] \right| \tag{3}
\end{aligned}$$

Let us denote assignments to vertices not in M_P collectively by $\vec{w} \in \{0, 1\}^{(k-2t)m}$, and assignments to vertices in M_P collectively by $\vec{u} \in \{0, 1\}^{2tm}$. Thus, using (2), we can continue (3) as follows:

$$\text{Disc}_{\mathcal{U}, P}(\text{IP}_G, R) \leq \mathbb{E}_{\vec{w}} \left[\left| \mathbb{E}_{\vec{u} \in \{0, 1\}^{2tm}} \left[\left(-1 \right)^{\sum_{(u_i, u_j) \in M_P} \langle \vec{u}_i, \vec{u}_j \rangle} R^{\vec{w}}(\vec{u}) \right] \right| \right] \tag{4}$$

Here $R^{\vec{w}}$ is a rectangle in $\{0, 1\}^{tm} \times \{0, 1\}^{tm}$ which is induced from R by fixing the assignments to \vec{w} . Observe when we fix the assignment to \vec{w} , the random variables \vec{u}_i also have same uniform distribution over $\{0, 1\}^m$ as \vec{u}_i . Hence the inner expectation is $\text{Disc}_{\mathcal{U}'}(\text{IP})$ where \mathcal{U}' is the uniform distribution over $\{0, 1\}^{tm} \times \{0, 1\}^{tm}$. The value of $\text{Disc}_{\mathcal{U}'}(\text{IP})$ is at most $2^{-\Omega(tm)}$ by Theorem 2.9. \blacktriangleleft

We will now consider a multi-party communication problem in the number-in-hand (NIH) model from the point of view of best partition communication complexity. While 2-party communication problems are relevant for proving lower bounds against circuits and ABP's, it'd be helpful to use the multi-party model for both unrestricted depth formulas and bounded-depth formulas. As the information bottleneck for players grows with the number of players in the NIH model, this kind of communication problems capture the limitation of formulas, especially w.r.t. ABPs and circuits.

► **Problem 3.8** (Communication problem Path-IP). *Let there be t players P_1, P_2, \dots, P_t . The problem is defined over a k vertex path Γ with a fixed vertex set $V = \{u_1, u_2, \dots, u_k\}$ and its partition into sets V_1, V_2, \dots, V_t . Each P_i gets the vertex set V_i . The input to this problem is a map $\pi_i : V_i \rightarrow \{0, 1\}^m$ given to each P_i which specifies a m bit vector assignment to each vertex in V_i . Together the players want to decide if $\langle \vec{i}_1, \vec{i}_2 \rangle + \langle \vec{i}_2, \vec{i}_3 \rangle + \dots + \langle \vec{i}_{k-1}, \vec{i}_k \rangle = 1 \pmod{2}$. Here for every $j \in [k]$ $\vec{i}_j = \pi_\ell(u_j)$ when $u_j \in V_\ell$ (for $\ell \in [t]$).*

Observe for this communication problem, rectangles are defined to be t -product sets, i.e. R is called a rectangle if $R = R_1 \times R_2 \times \dots \times R_t$ with each $R_i \subseteq \{0, 1\}^{k_i m}$ and $k_i = |V_i| \forall i \in [t]$. Our goal is to show under uniform distribution \mathcal{U} over $\{0, 1\}^{km}$, the discrepancy of any rectangle R is at most $2^{-\Omega(tm)}$. To show this we first show the following lemma.

► **Lemma 3.9.** *Consider a path graph Γ on k vertices. For every partition $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_t)$ of the vertex set of Γ into $2 \leq t < k$ parts, there exists a partition $\tilde{\mathcal{P}} = (\mathcal{P}_A, \mathcal{P}_B)$ into two parts that is a coarsening of \mathcal{P} such that $\tau(\Gamma, \tilde{\mathcal{P}})$ is $\Omega(t)$.*

Proof. Consider a partition $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_t)$ of the set of vertices $V(\Gamma)$. We create a random coarsening of it, $\tilde{\mathcal{P}} = (\mathcal{P}_A, \mathcal{P}_B)$, as follows: for every $i \in [t]$, toss an independent unbiased coin. If output is head, put the vertices of \mathcal{P}_i in \mathcal{P}_A and otherwise put them in \mathcal{P}_B . Since \mathcal{P} partitioned $V(\Gamma)$ into exactly t parts, there are at least $t - 1$ edges (u_i, u_{i+1}) of Γ such that u_i and u_{i+1} belong to different \mathcal{P}_j s. Denote by \mathcal{E} , the set of such edges. Now, for every edge $e = (u, v)$ in the path Γ define a random variable y_e such that

$$y_e = \begin{cases} 1 & \text{if } e \in \text{cut}(\tilde{\mathcal{P}}), \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\text{cut}(\tilde{\mathcal{P}})$ is the set of edges $e = (u, v)$ such that $u \in \tilde{\mathcal{P}}_A$ and $v \in \tilde{\mathcal{P}}_B$ or vice-versa. Let the random variable $Y = \sum_e y_e$ be the size of $\text{cut}(\tilde{\mathcal{P}})$. Note,

$$\mathbb{E}[Y] = \sum_e \mathbb{E}[y_e] \geq \sum_{e \in \mathcal{E}} \mathbb{E}[y_e] = \frac{t-1}{2} = \Omega(t).$$

Thus, there exists a fixed coarser partition $\tilde{\mathcal{P}} = (\mathcal{P}_A, \mathcal{P}_B)$ of $V(\Gamma)$ such that the induced bipartite graph $\Gamma_{\tilde{\mathcal{P}}}$ has $\Omega(t)$ many edges. Since Γ has maximum degree 2, by Lemma 3.3 $\tau(\Gamma, \tilde{\mathcal{P}}) = \Omega(t)$. ◀

► **Lemma 3.10.** *Let $t \geq 2$ and let Γ be a path on k vertices. Under the uniform distribution \mathcal{U} over $\{0, 1\}^{km}$ for any t -wise partition \mathcal{P} , the following holds: $\text{Disc}_{\mathcal{U}, \mathcal{P}}^t(\text{IP}_\Gamma) \leq 2^{-\Omega(tm)}$.*

Proof. Obtain a coarsening of the partition \mathcal{P} prescribed by Lemma 3.9 to get $\tilde{\mathcal{P}} = (\mathcal{P}_A, \mathcal{P}_B)$. Consider any rectangle $R = R_1 \times \dots \times R_t$ under partition \mathcal{P} . Let $A \subset [t]$ such that for every $i \in A$ the vertices of \mathcal{P}_i goes to \mathcal{P}_A . Similarly $B = [t] \setminus A$ and for every $j \in B$ vertices of \mathcal{P}_j goes to \mathcal{P}_B . Define, $R_A := \times_{i \in A} R_i$ and $R_B := \times_{j \in B} R_j$ and finally, $\tilde{R} := R_A \times R_B$. Observe that \tilde{R} forms a two-dimensional rectangle w.r.t. $\tilde{\mathcal{P}}$. It is simple to verify,

$$\text{Disc}_{\mathcal{U}}^t(R) = \text{Disc}_{\mathcal{U}}(\tilde{R}).$$

$$\text{We know using Lemma 3.7 } \text{Disc}_{\mathcal{U}, \tilde{\mathcal{P}}}(\tilde{R}) \leq \text{Disc}_{\mathcal{U}, \tilde{\mathcal{P}}}(\text{IP}_\Gamma) \leq 2^{-\Omega(tm)}. \quad \blacktriangleleft$$

4 Monotone Circuit Lower Bound via Expander Graph-IP Polynomial

In this section we prove Theorem 1.2. For the sake of convenience we restate it.

► **Theorem 1.2.** *Let G be a constant-degree expander graph on k vertices. Then, there exists a constant $c > 0$ such that any monotone circuit computing either of the polynomial $F_{k,n} \pm \epsilon \cdot f_{G,m}$ has size $2^{\Omega(km)}$ as long as $\epsilon \geq 2^{-ckm}$.*

Proof. Consider the Boolean function IP_G described in Subsection 3.1 where the underlying graph G is a constant degree expander graph on k vertices. Every vertex gets a m bit binary vector assignment. Let P be any nearly balanced partition of the vertex set V . We first show the following claim.

▷ **Claim 4.1.** Under the uniform distribution \mathcal{U} over $\{0,1\}^{km}$, for every nearly balanced partition P on the vertex set V , $\text{Disc}_{\mathcal{U},P}(\text{IP}_G) \leq 2^{-\Omega(km)}$.

Proof. From Lemma 3.4 we know that for every nearly balanced partition P of V , $\tau(G, P) = \Omega(k)$. Using Lemma 3.7, under the uniform distribution \mathcal{U} over $\{0,1\}^{km}$ we get that, $\text{Disc}_{\mathcal{U},P}(\text{IP}_G) \leq 2^{-\Omega(km)}$. ◀

Now the proof follows from the first part of Theorem 3.1. Here the polynomial $f = f_{G,m}$ and C^f is the Boolean function IP_G . From Claim 4.1 it is clear that

$$\gamma = \max_P \text{Disc}_{\mathcal{U},P}(\text{IP}_G) \leq 2^{-\Omega(km)}.$$

The universal distribution Δ is the uniform distribution \mathcal{U} over km bits. Let the value of $\gamma = 2^{-\gamma_0 km}$ for some constant $\gamma_0 > 0$. It is easy to verify that choosing $\epsilon \geq 2^{-\frac{\gamma_0 km}{10}}$ satisfies the condition $\epsilon \geq \frac{6\gamma}{1-3\gamma}$. Hence monotone circuit complexity of $F_{k,n} - \epsilon \cdot f_{G,m}$ is at least $\frac{\epsilon}{3\gamma}$ which is $2^{\Omega(km)}$. The proof for $F_{k,n} + \epsilon \cdot f_{G,m}$ is analogous. ◀

5 Separation between Monotone Circuits and Monotone ABPs via Tree-IP Polynomial

In this section we prove Theorem 1.4. For the sake of convenience we restate the theorem here.

► **Theorem 1.4.** *Let T be the full binary tree on k vertices. Then, $f_{T,m}$ can be computed by monotone circuits of size $O(kn^3)$. On the other hand, there exists a constant $c > 0$ such that any monotone ABP computing either of the polynomial $F_{k,n} \pm \epsilon \cdot f_{T,m}$ has size $k^{\Omega(m)}$ as long as $\epsilon \geq k^{-cm}$.*

The proof of this theorem is divided in the following two subsections.

5.1 Upper Bound

First we show the upper bound. Let T_u be the sub-tree rooted at node u of T . Below we consider set-multilinear monomials κ such that $I(\kappa) = V(T_u)$. Further, we denote by $\vec{\kappa}[u]$ the m -bit binary assignment to \vec{u} by $\vec{\kappa}$. For every node u in T and $\vec{a} \in \{0,1\}^m$ and $b \in \{0,1\}$, we define the following polynomials,

$$g_{u,b}^{\vec{a}} := \sum_{\substack{\text{IP}_{T_u}(\vec{\kappa}) = (-1)^b \\ \vec{\kappa}[u] = \vec{a}}} \kappa$$

and

$$g_{u,b} := \sum_{\vec{a} \in \{0,1\}^m} g_{u,b}^{\vec{a}}.$$

Thus, $g_{r,1}$ is the output polynomial $f_{T,m}$ where r is the root of T . By induction on the depth of T , we show that for each node u of T and $\vec{a} \in \{0,1\}^m$, the polynomials $g_{u,0}^{\vec{a}}, g_{u,1}^{\vec{a}}$ can be simultaneously computed by a circuit of size at most $O\left(2^d 2^{3m}\right)$.

For the base case $d = 1$. Let u be a node with two children v, w . For the purpose of this section, i, j, k are used for the integer values of $\vec{i}, \vec{j}, \vec{k} \in \{0,1\}^m$. Now the polynomials computed at node u are following,

$$g_{u,0}^{\vec{i}} = \sum_{\substack{\vec{j}, \vec{k} \in \{0,1\}^m: \\ \langle \vec{i}, \vec{j} \rangle + \langle \vec{i}, \vec{k} \rangle = 0 \pmod{2}}} x_{u,i} x_{v,j} x_{w,k}$$

where for every \vec{i} the polynomial $g_{u,0}^{\vec{i}}$ has 2^{2m} monomials. Hence we can compute the polynomials $g_{u,0}, g_{u,0}^{\vec{i}}$ by a monotone circuit of size at most 2^{3m} . Similarly we compute the polynomial $g_{u,1}$. So the total size of the circuit is 2^{3m+1} and the base case holds.

Consider a vertex u at depth d with children v, w . By inductive hypothesis, we have circuits C_v, C_w , each of size at most $O(2^{d-1} 2^{3m})$ computing simultaneously the polynomials $g_{v,0}^{\vec{i}}, g_{v,1}^{\vec{j}}$ and $g_{w,0}^{\vec{i}}, g_{w,1}^{\vec{j}}$ respectively, for each $\vec{i}, \vec{j} \in \{0,1\}^m$.

Now we compute the polynomials $g_{u,0}^{\vec{i}}, g_{u,1}^{\vec{j}}$.

It is easy to observe that $g_{u,0}^{\vec{i}} = Z_1 + Z_2$, where

$$Z_1 = \sum_{\substack{\vec{j}, \vec{k} \in \{0,1\}^m: \\ \langle \vec{i}, \vec{j} \rangle + \langle \vec{i}, \vec{k} \rangle = 0 \pmod{2}}} (x_{u,i} g_{v,0}^{\vec{j}} g_{w,0}^{\vec{k}} + x_{u,i} g_{v,1}^{\vec{j}} g_{w,1}^{\vec{k}})$$

and

$$Z_2 = \sum_{\substack{\vec{j}, \vec{k} \in \{0,1\}^m: \\ \langle \vec{i}, \vec{j} \rangle + \langle \vec{i}, \vec{k} \rangle = 1 \pmod{2}}} (x_{u,i} g_{v,0}^{\vec{j}} g_{w,1}^{\vec{k}} + x_{u,i} g_{v,1}^{\vec{j}} g_{w,0}^{\vec{k}})$$

Now from the circuits C_v and C_w we appropriately reuse the subcircuits for $\{g_{v,0}^{\vec{j}}, g_{v,1}^{\vec{j}}, g_{w,0}^{\vec{k}}, g_{w,1}^{\vec{k}}\}$. The other case, that of computing $g_{u,1}^{\vec{j}}$, is completely analogous.

Hence, the final circuit size, denoted by $S(d)$, satisfies the following recurrence:

$$S(d) \leq 2S(d-1) + O(2^{3m}).$$

Solving the recursion we get $S(d)$ is at most $O\left(2^d 2^{3m}\right)$. The upper bound follows since $k = 2^{d+1} - 1$.

5.2 Lower Bound

Next we show the lower bound result. Consider the Boolean function IP_T described in Subsection 3.1, where T is a full binary tree on k vertices.

▷ Claim 5.1. There exists a number $t \approx \frac{2}{3}k$ such that for any partition $P = (V_1, V_2)$ of the vertex set $V(T)$ with $|V_1| = t$, under the uniform distribution \mathcal{U} over $\{0, 1\}^{km}$,

$$\text{Disc}_{\mathcal{U}, P}(\text{IP}_T) \leq k^{-\Omega(m)}.$$

Proof. Using Lemma 3.5 we know there exists a number $t \approx \frac{2}{3}k$ such that for any partition $P = (V_1, V_2)$ with $|V_1| = t$, $\tau(T, P) = \Omega(\log k)$. By Lemma 3.7, under the uniform distribution \mathcal{U} over $\{0, 1\}^{km}$, we know that $\text{Disc}_{\mathcal{U}, P}(\text{IP}_T) \leq 2^{-\Omega(\tau(T, P) \cdot m)} = k^{-\Omega(m)}$. ◁

Now we apply the second part of Theorem 3.1 to prove the lower bound. Here the polynomial $f = f_{T, m}$ and C^f is the Boolean function IP_T . Using Claim 5.1,

$$\gamma = \max_P \text{Disc}_{\mathcal{U}, P}(\text{IP}_T) \leq k^{-\Omega(m)}.$$

Let $\gamma = k^{-cm}$ for some constant $c > 0$ and Δ be the uniform distribution \mathcal{U} over $\{0, 1\}^{km}$. One can easily verify that the condition $\epsilon \geq \frac{6\gamma}{1-3\gamma}$ is satisfied by choosing $\epsilon \geq k^{-\frac{cm}{10}}$. Using Theorem 3.1, the monotone ABP complexity of $g = F_{k, n} - \epsilon \cdot f_{T, m}$ is at least $\frac{\epsilon}{3\gamma}$ which is $k^{\Omega(m)}$. The proof for $F_{k, n} + \epsilon \cdot f_{T, m}$ is analogous.

References

- 1 Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing sets and an almost quadratic lower bound for syntactically multilinear arithmetic circuits. *Comb.*, 40(2):149–178, 2020. doi:10.1007/s00493-019-4009-0.
- 2 Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. On lower bounds for constant-width arithmetic circuits. In *20th International Symposium on Algorithms and Computation (ISAAC)*, pages 637–646. Springer-LNCS, 2009.
- 3 Vikraman Arvind and S. Raja. Some lower bound results for set-multilinear arithmetic computations. *Chic. J. Theor. Comput. Sci.*, 2016, 2016. URL: <http://cjtcs.cs.uchicago.edu/articles/2016/6/contents.html>.
- 4 Arkadev Chattopadhyay, Rajit Datta, Utsab Ghosal, and Partha Mukhopadhyay. Monotone complexity of spanning tree polynomial re-visited. 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 39:1–39:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.39.
- 5 Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. Lower bounds for monotone arithmetic circuits via communication complexity. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 786–799. ACM, 2021. doi:10.1145/3406325.3451069.
- 6 Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988. doi:10.1137/0217015.
- 7 Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. Separating multilinear branching programs and formulas. In *44th ACM Symposium on Theory of Computing (STOC)*, pages 615–624. ACM, 2012.
- 8 Thomas P. Hayes. Separating the k -party communication complexity hierarchy: An application of the zarankiewicz problem. *Discrete Mathematics & Theoretical Computer Science*, 13(4):15–22, 2011.
- 9 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.
- 10 Pavel Hrubes and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Comput. Complex.*, 20(3):559–578, 2011. doi:10.1007/s00037-011-0007-3.

- 11 Pavel Hrubes and Amir Yehudayoff. On isoperimetric profiles and computational complexity. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 89:1–89:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.89.
- 12 Pavel Hrubeš. On ϵ -sensitive monotone computations. *Computational Complexity*, 29(2):6, 2020. doi:10.1007/s00037-020-00196-6.
- 13 Balagopal Komarath, Anurag Pandey, and C.S. Rahul. Graph homomorphism polynomials: Algorithms and complexity. In *to appear in the 49th International Colloquium on Automata, Languages and Programming (ICALP)*, LIPICs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 14 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, USA, 2006.
- 15 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complex.*, 6(3):217–234, 1997. doi:10.1007/BF01294256.
- 16 Toniann Pitassi. Best-partition multiparty communication complexity. Manuscript online at <http://www.cs.toronto.edu/~toni/Courses/CommComplexity/Papers/bestpartition.ps>, 2009. Course notes for Foundations of Communication Complexity, Fall 2009.
- 17 Ramprasad Satharishi. A survey of lower bounds in arithmetic circuit complexity. Manuscript online at <https://github.com/dasarpmar/lowerbounds-survey/releases/download/v9.0.3/fancymain.pdf>, 2021. A selection of lower bounds in arithmetic circuit complexity.
- 18 Srikanth Srinivasan. Strongly exponential separation between monotone VP and monotone VNP. *ACM Trans. Comput. Theory*, 12(4):23:1–23:12, 2020. doi:10.1145/3417758.
- 19 Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.
- 20 Leslie G. Valiant. Negation is powerless for boolean slice functions. *SIAM J. Comput.*, 15(2):531–535, 1986. doi:10.1137/0215037.
- 21 Amir Yehudayoff. Separating monotone VP and VNP. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 425–429. ACM, 2019. doi:10.1145/3313276.3316311.

A Monotone Separations via Path-IP Polynomial

In this section, we prove Theorem 1.5 and Theorem 1.7.

A.1 Separation between Monotone ABPs and Monotone Formulas

For convenience Theorem 1.5 is restated below.

► **Theorem 1.5.** *Let Γ be a simple path on k vertices. Then, the polynomial $f_{\Gamma,m}$ can be computed by a monotone ABP of size $O(kn)$. On the other hand, there exists a constant $c > 0$ such that any monotone formula computing either of the polynomial $F_{k,n} \pm \epsilon \cdot f_{\Gamma,m}$ has size $k^{\Omega(m)}$ as long as $\epsilon \geq k^{-cm}$.*

The proof is divided in two parts.

Upper Bound

We first give the monotone ABP construction for the polynomial $f_{\Gamma,m}$. The ABP has $k+2$ layers $0, 1, \dots, k, k+1$. Layer 0 and $k+1$ are the source and sink vertex respectively. Let the path Γ be $u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_k$.

Layer 1 and k contains 2^m vertices labelled with $\{(u_1, i) | i \in [2^m]\}$ and $\{(u_k, j) | j \in [2^m]\}$ respectively. For every other layer $\ell \in [2, 3, \dots, k-1]$, we have 2^{m+1} vertices labelled with $\{(u_\ell, j)_b \mid j \in [2^m], b \in \{0, 1\}\}$. Next we describe the edge relations between consecutive layers.

- **Layer 0 to layer 1:** The source node s in layer 0 is connected to every node in layer 1. The edge label of $s \rightarrow (u_1, i)$ is labeled by variable $x_{u_1, i}$.
 - **Layer 1 to 2:** A node (u_1, i) is connected to $(u_2, j)_b$ in layer 2 if and only if $\langle \vec{i}, \vec{j} \rangle = b \pmod{2}$. The edge gets the label $x_{u_2, j}$. Here \vec{i}, \vec{j} are the binary representation of i and j respectively.
 - **Layer ℓ to $\ell+1$ for $\ell \in [2, k-2]$:** A node $(u_\ell, j)_b$ in layer ℓ is connected to the node $(u_{\ell+1}, j')_{b'}$ if and only if $b + \langle \vec{j}, \vec{j}' \rangle = b' \pmod{2}$. This edge label is $x_{u_{\ell+1}, j'}$.
 - **Layer $k-1$ to k :** A node $(u_{k-1}, i)_b$ is connected to the node (u_k, j) for if and only if $b + \langle \vec{i}, \vec{j} \rangle = 1 \pmod{2}$.
 - **Layer k to $k+1$:** Every node in layer k is connected to sink vertex with edge label 1.
- The size of the monotone ABP is $O(k2^m)$. Note that in the ABP construction each layer incrementally maintains the partial parity information. More precisely, a monomial $x_{u_1, i_1} x_{u_2, i_2} \dots x_{u_k, i_k}$ is generated exactly once between the source and sink if and only if $\text{IP}_\Gamma(\vec{i}_1, \dots, \vec{i}_k) = -1$. Hence the polynomial computed between the source and the sink is simply $f_{\Gamma,m}$.

Lower Bound

Consider the Boolean function IP_Γ described in Subsection 3.1, where Γ is a path on k vertices. We use the third part of Theorem 3.1 to prove the lower bound. Since there $\Omega(\log k)$ -wise partitions are considered, we set $t = \Omega(\log k)$. By Lemma 3.10, under the uniform distribution \mathcal{U} over $\{0, 1\}^{km}$ for every t -wise partition P we know that $\gamma = \text{Disc}_{\mathcal{U}, P}^t(\text{IP}_\Gamma) = 2^{-\Omega(tm)} = k^{-\Omega(m)}$.

Analogous to the case in Section 5, we now use the third part of Theorem 3.1 to show our lower bound against the size of monotone formulas computing $F_{k,n} - \epsilon \cdot f_{\Gamma,m}$ using the aboved discrepancy upper bound. As in Section 5, the lower bound follows by choosing $\epsilon \geq k^{-\Omega(m)}$ appropriately. The proof for $F_{k,n} + \epsilon \cdot f_{\Gamma,m}$ is analogous.

A.2 Separation between Monotone Constant Width ABPs and Monotone Constant Depth Formulas

Now we prove Theorem 1.7 which is restated below.

► **Theorem 1.7.** *Let Γ be a simple path on k vertices. Then, the polynomial $f_{\Gamma,1}$ can be computed by a monotone width-4 ABP of size $O(k)$. On the other hand, there exists a constant $c > 0$ such that any monotone formula of product-depth d computing either of the polynomial $F_{k,2} \pm \epsilon \cdot f_{\Gamma,1}$ has size $2^{\Omega(k^{1/d})}$ as long as $\epsilon \geq 2^{-ck^{1/d}}$.*

Upper Bound

Using the ABP construction given in Section A.1 we get a width-4 ABP of size $O(k)$ for the polynomial $f_{\Gamma,1}$.

Lower Bound

To show the lower bound, consider the Boolean function IP_Γ described in Subsection 3.1 where Γ is a k vertex path and each vertex gets a 1 bit assignment (i.e, $m = 1$). Using the fourth part of Theorem 3.1, we set $t = \Omega(k^{\frac{1}{d}})$. By Lemma 3.10 under the uniform distribution \mathcal{U} over $\{0, 1\}^k$ for every t -wise partition P we know $\text{Disc}_{\mathcal{U}, P}^t(\text{IP}_\Gamma) = 2^{-\Omega(k^{\frac{1}{d}})}$.

Now we use the fourth part of Theorem 3.1 to show monotone constant depth formula lower bound for polynomial $F_{k,n} - \epsilon \cdot f_{\Gamma,1}$ using the discrepancy upper bound. Similar to the earlier cases, the lower bound follows by choosing $\epsilon \geq 2^{-\Omega(k^{\frac{1}{d}})}$ appropriately.

B Structure Theorems for Monotone Set-Multilinear Formulas

The following theorems are re-statements of the corresponding theorems in [10]. There, they were stated for multilinear formulas. Here, we port the statements and their proofs from [10] to the set-multilinear setting needed for our work.

► **Theorem 2.6** ([10, Lemma 4]). *Let f be a degree k set-multilinear polynomial computed by a (monotone) formula of size s . Then there exists (monotone) set-multilinear-log-product polynomials $g_1, g_2, \dots, g_{s'}$ such that $s' \leq s$, $f = g_1 + g_2 + \dots + g_{s'}$.*

Proof. Let Φ be the (monotone) set-multilinear formula computing polynomial f . W.l.o.g Φ is a syntactically set-multilinear formula. For any node w in the formula let Φ_w be the sub-formula rooted at node w . Further we denote by $\Phi(w \leftarrow \beta)$, the formula obtained after removing the sub-formula rooted at node w and relabelling it by β . For any node w , let the polynomial computed at node w be f_w and $I(f_w)$ be the set of rows of matrix M on which the polynomial f_w is defined. First note the following claim.

▷ **Claim B.1.** There exists a node w in the formula Φ such that $\frac{k}{3} \leq |I(f_w)| \leq \frac{2k}{3}$.

Proof. W.l.o.g every node in Φ has in-degree 2. The polynomial computed at the root node r has $|I(f_r)| = k$. Now we traverse on the path towards the leaves by picking the child w among the children w, v when $|I(f_w)| \geq |I(f_v)|$. The traversing continues whenever the heavier child satisfies the condition $|I(f_w)| > \frac{2k}{3}$. We stop this process when we first encounter a node w such that $|I(f_w)| > \frac{2k}{3}$ but for its' children u, v $|I(f_u)|, |I(f_v)| \leq \frac{2k}{3}$. We will choose the heavier child here. I.e, we choose u if $|I(f_u)| \geq |I(f_v)|$ and the node u satisfies the property in the claim. ◁

We prove the theorem 2.6 by doing induction on s . For the base case when $s = 1$, the polynomial f is only one variable or constant. So, it trivially satisfies the conditions in the definition 2.5.

Using the claim B.1, let w be a node in the formula satisfying $\frac{k}{3} \leq |I(f_w)| \leq \frac{2k}{3}$ and size of $\Phi_w < s$. We can write the polynomial f in the following way,

$$f = g \cdot f_w + f'$$

where the polynomial f' is the set-multilinear polynomial computed by $\Phi(w \leftarrow 0)$. Clearly the formula size of Φ_w and $\Phi(w \leftarrow 0)$ is $< s$. Let they are s_w and $s(w \leftarrow 0)$. In particular $s_w + s(w \leftarrow 0) \leq s$. So we can use induction hypothesis on f_w and f' . That is, we can write $f_w = h_1 + \dots + h_{s'_w}$ and $f' = h'_1 + \dots + h'_{s'(w \leftarrow 0)}$ where for every i , h_i and h'_i are set-multilinear-log-product polynomials and $s'_w \leq s_w$, $s'(w \leftarrow 0) \leq s(w \leftarrow 0)$. Clearly $g \cdot h_i$ is set-multilinear polynomial and the sets $(I(g), I(h'_i))$ is a partition of $[k]$ where $\frac{k}{3} \leq |I(g)| \leq \frac{2k}{3}$. Hence the set-multilinear-log-product decomposition of f is

$$f = gh_1 + gh_2 + \dots + gh_{s'_w} + h'_1 + \dots + h'_{s'(w \leftarrow 0)}.$$

Next, we recall the structure theorem for monotone set-multilinear constant depth formulas.

► **Theorem 2.8** ([10, Lemma 9]). *Let f be a degree k set-multilinear polynomial computed by a (monotone) formula of size s and product depth d . Let $q > 1$ be a natural number such that $k > (2q)^d$. Then there exists (monotone) set-multilinear- $(q, k(2q)^{-d})$ -form polynomials $g_1, g_2, \dots, g_{s'}$ such that $s' \leq s$, $f = g_1 + g_2 + \dots + g_{s'}$.*

Proof. Let Ψ be the size s product depth d (monotone) set-multilinear formula computing f . For any node v in the formula we denote the sub-formula rooted at node v by Ψ_v . Further we define the polynomial computed at node v by f_v and $I(f_v)$ be the set of rows in the matrix $M_{k \times n}$ on which f_v is defined. First note the following claim.

▷ **Claim B.2.** Let $t > 1$ be any positive real number such that $k > t^d$. Then there exists a product node v in Ψ such that $|I(f_v)| \geq k \cdot t^{-d+1}$ and for every children u of v , $|I(f_u)| < \frac{|I(f_v)|}{t}$. Moreover if $t = 2q$ for $q \in \mathbb{N}$ then f_v is in $(q, k(2q)^{-d})$ -form.

Proof. The proof is by induction on product depth d .

For the base case $d = 1$. Let v be any product gate with children u_1, u_2, \dots, u_p . Clearly $|I(f_v)| = k$ and for every child, $|I(u_i)| \leq 1 < \frac{k}{t}$. So v is the required node in the claim.

Inductively assume the claim is true for every node at product depth $d' < d$. Let v be a product node at depth d with children u_1, \dots, u_p and $|I(f_v)| = k$. If for every children u_i , $|I(f_{u_i})| < \frac{|I(f_v)|}{t} = \frac{k}{t}$, then v is our required node. Otherwise, let u_i be a child such that $|I(f_{u_i})| \geq \frac{k}{t}$. Product depth of $u_i < d$. So by induction hypothesis there is a node w in Ψ_{u_i} at product depth $d' < d$, such that $|I(f_w)| \geq |I(f_{u_i})| \cdot t^{-d'+1} \geq k \cdot t^{-d+1}$. Also for every children w_i of w , $|I(f_{w_i})| < \frac{|I(f_w)|}{t}$. So, w is the required node for the claim.

Let v be the node in the claim with children u_1, \dots, u_p such that $|I(f_v)| = m \geq kt^{-d+1}$ and $|I(f_{u_i})| < \frac{m}{t}$. Then by appropriately grouping the polynomials f_{u_1}, \dots, f_{u_p} , it can be ensured that we get a new set of polynomials $\{g_1, g_2, \dots, g_{p'}\}$ such that $\frac{m}{t} \leq |I(g_j)| \leq \frac{2m}{t}$ for every $j \in [p']$. Hence f has $(\lfloor \frac{t}{2} \rfloor, \frac{m}{t})$ -form. Putting $t = 2q$ and $m = kt^{-d+1}$ we get our desired form. ◀

We prove the theorem 2.8 by doing induction on the size, s . The base case is easy to verify. By Claim B.2 there is a node v in the formula with polynomial f_v is in $(q, k(2q)^{-d})$ -form. Write

$$f = g \cdot f_v + f'$$

where f' is the polynomial computed by the formula Ψ after removing the sub-formula rooted at node v and relabelling it by the element 0. Clearly the sets $I(g)$ and $I(f_v)$ forms a partition of the rows of matrix. From Claim B.2 The polynomial f_v is in $(q, k(2q)^{-d})$ form. That is $f_v = f_1 \cdot f_2 \cdots f_q$ where each $|I(f_i)| \geq k(2q)^{-d}$. Clearly the polynomial $(gf_1) \cdots f_q$ is also in $(q, k(2q)^{-d})$ -form. Hence the proof follows by doing induction on the sub-formula of size $< s$ computing f' . ◀

C Good Matching in Constant Degree Expander Graphs


► **Lemma 3.4** ([16, 8]). *Let d be a constant and G be a d regular expander graph on k vertices with the second largest eigen value of the normalized adjacency matrix $\lesssim \frac{1}{\sqrt{d}}$ and $P = (V_1, V_2)$ be a nearly balanced partition of the vertex set V . Then, $\tau(G, P) = \Omega_d(k)$.*

Proof. Take the partition $P = (V_1, V_2)$ such that $\frac{k}{3} \leq |V_1|, |V_2| \leq \frac{2k}{3}$ and construct the induced bipartite graph G_P . using Expander Mixing Lemma [9, Lemma 2.5] we know $|E(G_P)| \geq \frac{d|V_1||V_2|}{k} - \lambda d \sqrt{|V_1||V_2|}$. Substituting the values of $\lambda, |V_1|$ and $|V_2|$ we get $|E(G_P)| = \Omega_d(k)$. Since it is a constant degree regular graph, using Lemma 3.2 we get $\tau(G, P) = \Omega_d(k)$. ◀

Inscribing or Circumscribing a Histogram to a Convex Polygon

Jaehoon Chung ✉

Department of Computer Science and Engineering, Pohang University of Science and Technology, Korea

Sang Won Bae ✉ 


Division of Computer Science and Engineering, Kyonggi University, Suwon, Korea

Chan-Su Shin ✉ 

Division of Computer Engineering, Hankuk University of Foreign Studies, Seoul, Korea

Sang Duk Yoon ✉ 

Department of Service and Design Engineering, SungShin Women's University, Seoul, Korea

Hee-Kap Ahn ✉ 

Graduate School of Artificial Intelligence, Department of Computer Science and Engineering, Pohang University of Science and Technology, Korea

Abstract

We consider two optimization problems of approximating a convex polygon, one by a largest inscribed histogram and the other by a smallest circumscribed histogram. An axis-aligned histogram is an axis-aligned rectilinear polygon such that every horizontal edge has an integer length. A histogram of orientation θ is a copy of an axis-aligned histogram rotated by θ in counterclockwise direction. The goal is to find a largest inscribed histogram and a smallest circumscribed histogram over all orientations in $[0, \pi)$. Depending on whether the horizontal width of a histogram is predetermined or not, we consider several different versions of the problem and present exact algorithms. These optimization problems belong to shape analysis, classification, and simplification, and they have applications in various cost-optimization problems.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Shape simplification, Shape analysis, Histogram, Convex polygon

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.13

Funding This research was partly supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)).

Sang Won Bae: supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07042755).

Chan-Su Shin: supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1F1A1058963).

Sang Duk Yoon: supported by “Cooperative Research Program for Agriculture Science & Technology Development (Project No. PJ015269032022)” Rural Development Administration, Republic of Korea.

1 Introduction

We consider two optimization problems of approximating a convex polygon, one by a largest inscribed histogram and the other by a smallest circumscribed histogram. An axis-aligned *histogram* is an axis-aligned rectilinear (possibly weakly simple) polygon such that every



© Jaehoon Chung, Sang Won Bae, Chan-Su Shin, Sang Duk Yoon, and Hee-Kap Ahn; licensed under Creative Commons License CC-BY 4.0

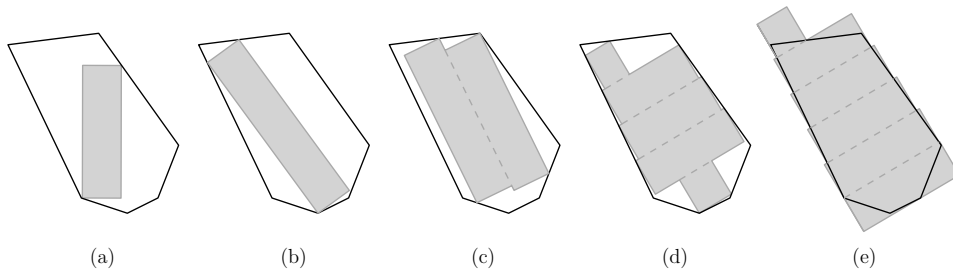
42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 13; pp. 13:1–13:16



Leibniz International Proceedings in Informatics

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



■ **Figure 1** (a) The largest axis-aligned inscribed unit histogram. (b) The largest inscribed unit histogram. (c) The largest inscribed 2-histogram. (d) The largest inscribed histogram. (e) The smallest circumscribed histogram.

horizontal edge has an integer length. We call an axis-aligned histogram of width 1 an axis-aligned *unit histogram* and an axis-aligned histogram of *width* k an axis-aligned k -*histogon*. Thus, an axis-aligned unit histogram is simply an axis-aligned rectangle of horizontal width 1, and its height is the length of the vertical sides. An axis-aligned k -histogon for a positive integer k can be described by k axis-aligned and interior-disjoint unit histograms. A histogram of a fixed orientation $\theta \in [0, \pi)$ is a copy of an axis-aligned histogram rotated by θ in counterclockwise direction.

In the inscribed histogram problem, we compute a histogram with maximum area that can be inscribed in P . We call such a histogram a *largest* inscribed histogram of P . Depending on whether the horizontal width of a histogram is predetermined (1 or a positive integer k) or not, we consider three versions of the problem. In the circumscribed histogram problem, we compute a histogram with minimum area that can be circumscribed to P . We call such a histogram a *smallest* circumscribed histogram of P . See Figure 1 for an illustration.

The optimization problems we investigate belong to shape analysis, classification, and simplification [2, 3]. Many optimization problems occurred in those research topics are concerned with the largest inscribed figure and the smallest circumscribed figure of a prescribed shape. The largest inscribed histogram problem and the smallest circumscribed histogram problem have applications in several coverage path planning (CPP) problems [7, 11, 15, 16] such as mowing a lawn using a mower, painting a piece using a spray gun, inspecting the surface of an object by a scanner, and milling a pocket by moving a cutter. Other applications include several topics in calculus, including Riemann sums and optimization. For a function graph (or a curve), the area under the graph can be approximated by a histogram: an inscribed histogram is an under-approximation of the area and a circumscribed histogram is an over-approximation of the area.

Related Work. There has been a lot of work in approximating shapes in past decades. The goal is to find a polygon inscribing or circumscribing another polygon (a convex polygon or a simple polygon) while maximizing (or minimizing) a certain measure. There are algorithms for finding triangles with maximum area or maximum perimeter inscribed in a convex polygon and a simple polygon [20, 24]. A convex k -gon with maximum area or maximum perimeter inscribed in a convex n -gon can be computed in $O(kn + n \log n)$ time [1, 24]. Chang et al. [8] gave an $O(n^7)$ -time algorithm for finding a convex polygon with maximum area and an $O(n^6)$ -time algorithm for finding a convex polygon with maximum perimeter inscribed in a convex n -gon. There are $O(n)$ -time algorithms for finding triangles with minimum area or minimum-perimeter circumscribing a convex n -gon [5, 21].

DePano et al. [14] gave algorithms for finding an equilateral triangle with maximum area and a square with maximum area inscribed in a polygon either convex or simple. Lee et al. [19] gave algorithms for finding maximum-area triangles with fixed interior angles inscribed

in a polygon, either convex, simple, or even non-simple possibly with holes. Intensive research has been done for the problems of finding rectangles with maximum area or maximum perimeter inscribed in a convex polygon and a simple polygon [6, 10, 18]. Jin et al. [17] gave an $O(n^2)$ -time algorithm for computing all parallelograms with maximum area in a convex n -gon. Toussaint [23] gave an $O(n)$ -time algorithm for finding rectangles with minimum area or minimum perimeter circumscribing a convex n -gon. Schwarz et al. [22] gave a simple $O(n)$ -time algorithm for finding a parallelogram with minimum area in a convex n -gon.

Very recently, the authors [12] presented first algorithms for the axis-aligned case of our histogram problem for a convex polygon P . These previous algorithms include: an $O(\log n)$ -time algorithm for a largest axis-aligned inscribed unit histogram, an $O(\min\{n, k \log^2 \frac{n}{k}\})$ -time algorithm for a largest axis-aligned inscribed k -histogon for a fixed $k > 1$, an $O(\min\{n, w \log^2 \frac{n}{w}\})$ -time algorithm for a largest axis-aligned inscribed histogram, where w denotes the width of a largest axis-aligned histogram inscribed in P , and an $O(\min\{n, W \log \frac{n}{W}\})$ -time algorithm for a smallest axis-aligned circumscribed histogram, where W denotes the horizontal width of P .

Our Results. For the problem of inscribing a largest unit histogram in a convex n -gon, we present an $O(n \log n + U)$ -time algorithm using $O(n)$ space, where U denotes the total number of intersections between unit circles centered at vertices of P and the edges of P . In the worst case, the quantity U can be quadratic in n , while it is near-linear in most cases. In addition, we present another algorithm that runs in $O(n \log n + n/\delta)$ time under the assumption that there exists a unit histogram of height at least δ in P where $0 < \delta < 1$. This provides a faster way once one asserts the existence of any unit histogram of positive height contained in P , by any means such as efficient approximation algorithms. We also show that our algorithm can determine whether there exists a unit histogram of height δ in P in the same time bound for any input $0 < \delta < 1$. Based on these results, a largest inscribed unit histogram can be computed more efficiently: in $O(n \log n)$ time if its height h is $\Omega(1/\log n)$, or in $O(n \log n \log \log \frac{1}{h \log n} + n/h^2)$ time in an output-sensitive way, otherwise.

We also present an algorithm that, given a positive integer k , finds a largest k -histogon inscribed in P in $O(kn^2(\log n + kT(\min\{k, n\})))$ time using $O(\min\{k, n\}n)$ space, where $T(m)$ denotes the time complexity of the optimization step for the trigonometric expression with $O(m)$ terms, each of quadratic form. For finding a largest histogram with no restriction on the width inscribed in P , we present an $O(Dn^2(\log n + T(\min\{D, n\})))$ -time algorithm using $O(\min\{D, n\}n)$ space, where D denotes the diameter of P . Barequet and Rogol [4] observed that $T(m) = O(m)$ in practice.

Finally, for finding a smallest circumscribed histogram of a convex n -gon, we present an $O(Dn(\log(\min\{D, n\}) + T(\min\{D, n\})))$ -time algorithm using $O(n)$ space.

Due to lack of space, some proofs and details are omitted. They can be found in the full version of the paper.

2 Preliminaries

Let P be a convex polygon with n vertices, given in a list sorted in counterclockwise order along the boundary of P . For ease of discussion, we assume that no two edges of P are parallel to each other.

For a connected set X , we denote by ∂X the boundary of X , by $\text{int}(X)$ the interior of X , and by $\text{cl}(X)$ the closure of X . For a point $p \in \mathbb{R}^2$, let $x(p)$ and $y(p)$ be the x -coordinate and the y -coordinate of p , respectively. For any two points p and q in \mathbb{R}^2 , we use pq to

13:4 Inscribing or Circumscribing a Histogram to a Convex Polygon

denote the line segment connecting p and q , and by $|pq|$ the length of pq . If both p and q lie on ∂P , pq is called a *chord* of P . A chord of unit length in P is called a *unit chord* of P . We use $\partial P[x, y]$ to denote the portion of ∂P from x to y in counterclockwise order, and let $\partial P(x, y) = \partial P[x, y] \setminus \{x\}$, $\partial P[x, y) = \partial P[x, y] \setminus \{y\}$, and $\partial P(x, y) = \partial P[x, y] \setminus \{x, y\}$. We use $P[x, y]$ to denote the subpolygon of P enclosed by $\partial P[x, y]$ and the chord xy .

A histogram of a fixed orientation $\theta \in [0, \pi)$ is a copy of an axis-aligned histogram rotated by θ in counterclockwise direction. The width of a histogram H of orientation θ is the width of the axis-aligned copy of H that is obtained by rotating H by θ in clockwise direction. Let $w(H)$ be the width of H and $|H|$ denote the area of H .

The *orientation* of a line is the angle swept from the x -axis in a counterclockwise direction to the line, and it is thus in $[0, \pi)$. The *orientation* of a line segment is that of the line containing it. We mean by a *boundary element* of P its vertex or edge. For each $\theta \in [0, \pi)$, there are exactly two lines of orientation θ that are tangent to P . Each of these two tangent lines intersects ∂P in a boundary element of P . We call a pair (m_1, m_2) of boundary elements of P *antipodal* if there exists an orientation $\theta \in [0, \pi)$ of which two tangent lines intersect P in m_1 and m_2 . Toussaint [23] showed the following, introducing the rotating caliper.

► **Lemma 1** (Toussaint [23]). *There are $O(n)$ antipodal pairs of a convex n -gon P , and they can be computed in $O(n)$ time.*

Note that if (m_1, m_2) is an antipodal pair, then not both of m_1 and m_2 can be edges of P , since P has no two parallel edges.

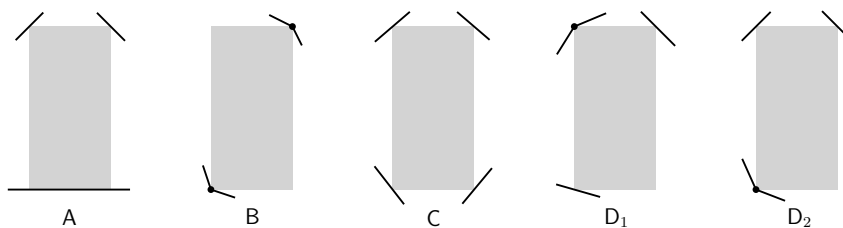
3 Largest inscribed unit histogram

In this section we compute a largest inscribed unit histogram of P . Cabello et al. [6] gave an algorithm that computes a largest inscribed rectangle in a convex polygon with n vertices in $O(n^3)$ time. They showed that the set of parallelograms contained in a convex polygon with n vertices can be parameterized by a convex polytope in \mathbb{R}^6 defined by $4n$ linear constraints. Their algorithm triangulates the boundary of the convex polytope and finds a largest rectangle for each simplex of the triangulation. Since the optimization problem for a simplex has a constant size and the complexity of the convex polytope is $O(n^3)$, their algorithm takes $O(n^3)$ time.

We can modify their algorithm so that it works for our problem. For each of $O(n^3)$ optimization problems, we add a new non-linear constraint that a side of parallelogram has length 1. Since the size of altered optimization problem is constant, we can find a largest inscribed unit histogram of P in $O(n^3)$ time.

In the following, we present an algorithm for the problem that runs in $O(n \log n + U)$ time, where U denotes the total number of intersections between unit circles centered at vertices of P and the edges of P . Since a unit circle intersects an edge of P at most twice and it may intersect $O(n)$ edges of P , we have $U = O(n^2)$. Indeed, we can construct a convex polygon such that $U = \Omega(n^2)$. In practical situations, however, most unit circles centered at vertices intersect only few edges of P , so the total number of intersections is either linear or near-linear and our algorithm runs faster.

A largest inscribed unit histogram of P touches some boundary elements (edges and vertices) of P . For a unit histogram \bar{H} , we say that there is a *side-contact* if a side of \bar{H} is fully contained in an edge of P , or a *corner-contact* if a corner of \bar{H} lies on ∂P . The *contact set* of \bar{H} is the set of all of its corner-contacts and side-contacts. We show that the contact set of any largest inscribed unit histogram falls into one of the following four types. See Figure 2.



■ **Figure 2** Types of contact sets of for largest inscribed unit histogons of P .

► **Lemma 2.** *For any largest inscribed unit histogon of P , its contact set satisfies one of the following conditions:*

- A. *It has a side-contact.*
- B. *It consists of two corner-contacts at two vertices of P that are antipodal.*
- C. *It consists of four corner-contacts.*
- D. *It consists of three corner-contacts, one of which is associated with a vertex of P .*

By Lemma 2, our algorithm finds a largest unit histogon contained in P whose contact set falls into each of the four types.

3.1 One side-contact (type A) or two corner-contacts (type B)

Any unit histogon of type A contained in P has a side-contact with an edge of P , so a side of such a histogon has the same orientation with an edge of P . This reduces the problem to its fixed-orientation variant, which can be solved in $O(\log n)$ time [12]. Hence, a largest inscribed unit histogon of type A can be found in $O(n \log n)$ time by solving $O(n)$ instances of the fixed-orientation problem.

For unit histogons of type B, we first specify all antipodal pairs of P by Lemma 1, and consider each antipodal pair (v_1, v_2) such that both v_1 and v_2 are vertices of P . There are at most two possible unit histogons H such that v_1 and v_2 are opposite corners of H . We try each of these two unit histogons and test if it is contained in P . This containment test can be done in $O(\log n)$ time [13]. Since there are $O(n)$ antipodal pairs by Lemma 1, we can find a largest inscribed unit histogon of type B in $O(n \log n)$ time.

► **Lemma 3.** *We can compute a largest inscribed unit histogon of P that has a side-contact (type A) or two corner-contacts (type B) in $O(n \log n)$ time.*

3.2 Four corner-contacts (type C)

A unit histogon of type C has top and bottom sides lying on two parallel unit chords of P by Lemma 2. We say two parallel unit chords of P are *aligned orthogonally* if their convex hull forms a unit histogon. A largest inscribed unit histogon of type C in P has a positive height if and only if there are the two distinct parallel unit chords of P that are aligned orthogonally.

We find all inscribed unit histogons of type C in P and return the largest one among them. To do this, we trace two unit chords of orientation θ in P while θ increasing from 0 to π , and find the orientations at which the two unit chords are aligned orthogonally. The following lemma is about the existence of two distinct parallel unit chords in P .

► **Lemma 4.** *The following statements are equivalent:*

- *The length of a longest chord of orientation θ in P is larger than 1.*
- *Either there are exactly two distinct unit chords of orientation θ , or P has an edge of orientation θ with length larger than 1.*

13:6 Inscribing or Circumscribing a Histogram to a Convex Polygon

Let O denote the set of all orientations $\theta \in [0, \pi)$ such that the longest chord of orientation θ in P has length larger than 1, but there is no edge of P of orientation θ whose length is larger than 1. By Lemma 4, there are exactly two distinct unit chords of orientation θ in P if and only if $\theta \in O$. Note that O is our search space for all possible unit histograms of type C. In the following, we investigate the motion of the longest chord and unit chords of orientation θ as θ continuously increases.

► **Lemma 5.** *Both endpoints of the longest chord of orientation θ in P move monotonously in the counterclockwise direction along ∂P as θ increases. Moreover, the pair of two boundary elements of P on which the endpoints of the longest chord lie is antipodal.*

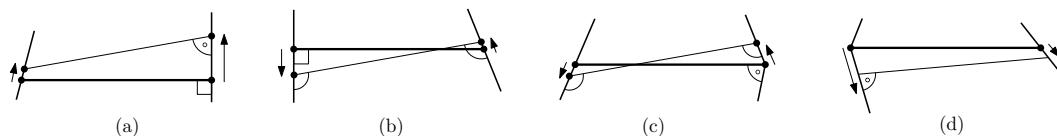
For any orientation $\theta \in [0, \pi)$, consider the longest chord ab of orientation θ and the two boundary elements e_a, e_b of P such that $a \in e_a$ and $b \in e_b$. It is obvious that the longest chord continuously moves as θ increases while its endpoints lie on e_a and e_b , respectively, unless both e_a and e_b are vertices. This implies that the length of the longest chord also changes continuously locally and thus that the set O forms several open intervals of orientations. Let $\mathcal{I}(O)$ be the set of these intervals induced by O .

By Lemma 5, together with Lemma 1, there are only $O(n)$ different pairs of such boundary elements (e_a, e_b) that the endpoints of the longest chord may land on. Hence, the set $\mathcal{I}(O)$ consists of $O(n)$ open intervals. These intervals can be computed in $O(n)$ time by processing each antipodal pair in $O(1)$ time after specifying them explicitly in $O(n)$ time by Lemma 1.

► **Lemma 6.** *Both endpoints of a unit chord of orientation θ move continuously along ∂P as θ continuously increases over any interval in $\mathcal{I}(O)$.*

We define the combinatorial structure of a unit chord to be the pair of the boundary elements of P where its endpoints lie. The combinatorial structures of unit chords of orientation θ may change for θ increasing in an interval of $\mathcal{I}(O)$. Since the endpoints of a unit chord move continuously along ∂P by Lemma 6, such a change occurs only if an endpoint of the unit chord meets a vertex of P . We call an orientation at which the combinatorial structure of a unit chord changes a *v-event* orientation. The set of v-event orientations partitions the intervals of $\mathcal{I}(O)$ into subintervals, called *v-intervals*. Note that, for any v-interval I , the combinatorial structure of the unit chords of orientation θ remains the same over all $\theta \in I$. Moreover, the number of v-intervals of $\mathcal{I}(O)$ is bounded by $O(n + U)$ since each v-event corresponds to an intersection between a unit disk centered at a vertex of P and an edge of P .

Now, we compute v-event orientations, v-intervals, and unit histograms of type C by processing the intervals of $\mathcal{I}(O)$, one by one, in the increasing order of orientation as follows. Consider an interval $I = (\theta_0, \theta_1) \in \mathcal{I}(O)$, and assume that we have processed all intervals prior to I and are now about to process I . By definition of O , note that $\theta_0 \notin O$. This implies that either there is an edge e_0 of P of orientation θ_0 whose length is larger than 1, or the longest chord of orientation θ_0 has length at most 1. For the former case, there is the previous interval $I' \in \mathcal{I}(O)$ that share the endpoint θ_0 with I , and thus we have the unit chord of orientation θ_0 not lying on e_0 when we process I . For the latter case, the length of the longest chord of orientation θ_0 is exactly 1 by Lemma 6. Thus, the longest chord is the only unit chord of orientation θ_0 and it can be specified when we compute $\mathcal{I}(O)$. As θ increases from θ_0 to θ_1 over I , the endpoints of the unit chords of orientation θ move continuously along ∂P for θ increasing in I by Lemma 6. So, we can trace the endpoints of the unit chords of orientation θ and compute v-event orientations in I and v-intervals induced by the orientations in order.



■ **Figure 3** Two edge events and two vertex events where the feasibility of a top unit chord C (thick segment) changes for θ increasing from θ_1 in the interior of an interval of $\mathcal{I}(O)$. Assume $\theta_1 = 0$. Each acute angle is marked with a circle. (a) An edge event where C becomes infeasible. (b) An edge event where C becomes feasible. (c) A vertex event where C becomes feasible. (d) a vertex event where C becomes infeasible.

For a v -interval I' determined during this tracing, we check whether there is an orientation $\theta \in I'$ such that the two unit chords are aligned orthogonally, that is, the convex hull of the two unit chords of orientation θ forms a unit histogram. Since the combinatorial structures of the unit chords does not change in the v -interval, this can be done by solving an equation defined for $\theta \in I'$ such that the equation has a solution if and only if the two unit chords are aligned orthogonally. From the edges of P corresponding to the combinatorial structure of a unit chord of orientation $\theta \in I$, we can express the x -coordinate of an endpoint of the chord as $A \sin(\theta + B) + C$ using the law of sines, where A , B and C are constants. The equation has $O(1)$ solutions which can be found in $O(1)$ time.

To sum up, we compute all v -event orientations and v -intervals in $O(n + U)$ time. For each of the $O(n + U)$ v -intervals, we determine whether there is a unit histogram of type C of an orientation in the v -interval in $O(1)$ time. Thus, we compute all unit histograms of type C in $O(n + U)$ time.

► **Lemma 7.** *We can compute the largest inscribed unit histogram of P that has four corner-contacts (type C) in $O(n + U)$ time using $O(n)$ space.*

3.3 Three corner-contacts (type D)

We denote by \bar{H}^* the largest inscribed unit histogram of type D in P and let θ^* be the orientation of its top and bottom sides. By Lemma 2, one corner of \bar{H}^* lies at a vertex of P and the top or bottom side of \bar{H}^* are unit chords of orientation θ^* in P .

We say a unit chord C of orientation $\theta \in O$ is *feasible* if there exists a unit histogram \bar{H} in P of a positive height such that C is a top or bottom side of \bar{H} . If C is the top side (or the bottom side, respectively) of \bar{H} , we call C a *top* (or a *bottom*, respectively) unit chord and the orientation θ *top feasible* (or *bottom feasible*, respectively).

In the following, we consider the case that the top side of \bar{H}^* is a top feasible unit chord and the bottom-left corner of \bar{H}^* lies on ∂P . The other cases in which the bottom-right corner of \bar{H}^* lies on ∂P or the bottom side of \bar{H}^* is a bottom feasible unit chord can be handled analogously. Our strategy is to trace the top feasible unit chord of orientation θ while θ increases in each interval of $\mathcal{I}(O)$ and to find all unit histograms inscribed in P whose top side coincides with a top unit chord.

Events and event orientations. As θ continuously increases over an interval $I \in \mathcal{I}(O)$, a unit chord of orientation θ becomes feasible and infeasible at certain orientations in I . We call each such orientation an *f-event* orientation. There are two types of *f-event* orientations of a unit chord C . See Figure 3 for an illustration.

- *Edge event:* C is orthogonal to an edge on which an endpoint of C lies.
- *Vertex event:* An endpoint of C meets a vertex and both interior angles at the vertex of the two subpolygons of P induced by C are acute.

Then there is some $\zeta > 0$ such that either C is infeasible at $\theta - \epsilon$ but feasible at $\theta + \epsilon$ for any $0 < \epsilon \leq \zeta$, or C is feasible at $\theta - \epsilon$ but infeasible at $\theta + \epsilon$ for any $0 < \epsilon \leq \zeta$.

Note that we can determine whether a unit chord is at f-event or v-event (defined in Section 3.2) while tracing the endpoints of the unit chord. Thus, we can compute all f-event orientations along with v-event orientations while tracing the two unit chords.

For a top feasible orientation θ , we denote by $\alpha(\theta)$ the top unit chord of orientation θ and let $\bar{H}(\theta)$ be the largest unit histogram inscribed in P whose top side is $\alpha(\theta)$. Let $\alpha_1(\theta)$ and $\alpha_2(\theta)$ be the two endpoints of $\alpha(\theta)$ that correspond to the top-right corner and the top-left corner of $\bar{H}(\theta)$, respectively.

By Lemma 6, $\alpha_1(\theta)$ and $\alpha_2(\theta)$ move continuously along ∂P as θ continuously increases over the interval. Since P is convex, the endpoints of the two chords orthogonal to $\alpha(\theta)$ through $\alpha_1(\theta)$ and $\alpha_2(\theta)$ also move continuously along ∂P . Observe that $\bar{H}(\theta)$ always has at least one bottom corner at one endpoint of the orthogonal chords. When $\bar{H}(\theta)$ has both bottom corners at endpoints of the orthogonal chords, it has four corner-contacts (type C), and $\bar{H}(\theta - \epsilon)$ has its bottom-left corner lying on ∂P and $\bar{H}(\theta + \epsilon)$ has its bottom-right corner lying on ∂P (or in the opposite way) for sufficiently small ϵ .

Intervals containing no event orientations. We partition the intervals of $\mathcal{I}(O)$ by the f-event orientations, v-event orientations, and the orientations where unit histograms of type C are defined. Then we gather (partitioned) intervals I such that for any orientation $\theta \in I$, θ is a top feasible orientation and $\bar{H}(\theta)$ has its bottom-left corner $q(\theta)$ on ∂P . The resulting set of intervals is denoted by \mathcal{I}_L . The number of intervals in \mathcal{I}_L is $O(n + U)$, and for an interval $I \in \mathcal{I}_L$, the combinatorial structure of $\alpha(\theta)$ remains the same for any orientation $\theta \in I$.

Let u denote the topmost vertex of P . If there are more than one topmost vertex, let u be the one with the largest x -coordinate. We claim that for any $I \in \mathcal{I}_L$ and $\theta \in I$, u , $\alpha_2(\theta)$ and $q(\theta)$ appear in counterclockwise order along ∂P . When $\theta = 0$, it clearly holds. Suppose that u lies on $\partial P[\alpha_2(\theta), q(\theta)]$ for some $\theta > 0$. Since both interior angles, one at $\alpha_2(\theta)$ and one at $q(\theta)$, in $P[\alpha_2(\theta), q(\theta)]$ are smaller than $\pi/2$, the line of orientation θ passing through u intersects $\partial P(q(\theta), \alpha_2(\theta))$ at a boundary point with y -coordinate larger than that of u , a contradiction.

► **Lemma 8.** *For any $I_1, I_2 \in \mathcal{I}_L$ and any two orientations $\theta_1 \in I_1$ and $\theta_2 \in I_2$ with $\theta_1 < \theta_2$, if $q(\theta_2) \in \partial P(u, q(\theta_1))$, then $|\bar{H}(\theta_2)| < |\bar{H}^*|$.*

For any unit histogram \bar{H} of type D, we distinguish two further subcases: \bar{H} is of type D_1 if a corner incident to its top side lies on a vertex of P , or of type D_2 if its bottom-left corner lies on a vertex of P . For an illustration, see Figure 2. If \bar{H}^* is of type D_1 , θ^* is an endpoint of an interval of \mathcal{I}_L by definition of v-event in Section 3.2. If \bar{H}^* is of type D_2 , $q(\theta^*)$ lies on a vertex of P . By Lemma 8, there is no $\theta \in I$ with $I \in \mathcal{I}_L$ such that $\theta < \theta^*$ and $q(\theta^*) \in \partial P(u, q(\theta))$.

Algorithm. Our algorithm processes the intervals of \mathcal{I}_L one by one in increasing order of orientation, and computes \bar{H}^* as follows. It maintains a point w lying on ∂P to indicate the portion $\partial P[u, w]$ that has been processed so far. In the beginning, w is set to the topmost vertex u . Let $v(w)$ denote the counterclockwise neighbor vertex of w . The algorithm processes an interval $I \in \mathcal{I}_L$ as follows. Let θ_0 and θ_1 be the endpoints of I with $\theta_0 < \theta_1$. A largest inscribed unit histogram of type D_1 is computed in Step 1 and Step 3, and a largest inscribed unit histogram of type D_2 is computed in Step 2.

- Step 1.** If θ_0 is a top feasible orientation and $q(\theta_0) \in \partial P[w, u]$, compute $\bar{H}(\theta_0)$ and update w to $q(\theta_0)$.
- Step 2.** Repeat the following if there is $\theta \in [\theta_0, \theta_1]$ such that $q(\theta) \in \partial P[v(w), u]$.
- a. Find the smallest $\theta' \in [\theta_0, \theta_1]$ such that $v(w)$ is met by $q(\theta')$, and compute $\bar{H}(\theta')$ if such θ' exists.
 - b. Update w to $v(w)$.
- Step 3.** If θ_1 is a top feasible orientation and $q(\theta_1) \in \partial P[w, u]$, compute $\bar{H}(\theta_1)$ and update w to $q(\theta_1)$.

► **Lemma 9.** *Our algorithm computes \bar{H}^* while processing the interval I of \mathcal{I}_L with $\theta^* \in I$.*

Analysis. Now we analyze the running time of the algorithm. The algorithm processes the intervals in \mathcal{I}_L in the increasing order, one by one. Recall that it computes f-event orientations, v-event orientations, and orientations of unit histogons of type C while tracing the unit chords of orientation θ as θ runs over each interval in $\mathcal{I}(O)$. They can be found in the increasing order of orientation in $O(n + U)$ time using $O(n)$ space.

In Step 1, the algorithm checks whether $q(\theta_0) \in \partial P[w, u]$ or not in $O(1)$ time using $\alpha(\theta_0)$ and w . If $q(\theta_0) \in \partial P[w, u]$, it finds the edge where $q(\theta_0)$ lies by checking the edges of P one by one in counterclockwise order from the edge where w lies, and updates w to $q(\theta_0)$. Thus, Step 1 is done in time linear to the number of edges checked for updating w . Similarly, Step 3 can be done in the same time.

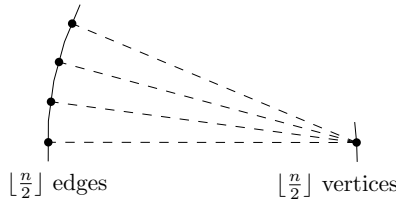
Now consider Step 2. The algorithm checks if there is an orientation $\theta \in [\theta_0, \theta_1]$ such that $q(\theta) \in \partial P[v(w), u]$. This can be done in $O(1)$ time as the combinatorial structure of $\alpha(\theta)$ has size $O(1)$ for $\theta \in [\theta_0, \theta_1]$. If there exists such an orientation, the algorithm finds the smallest $\theta' \in [\theta_0, \theta_1]$ such that $v(w)$ is met by $q(\theta')$, and updates w to $v(w)$. This also takes $O(1)$ time as the corresponding combinatorial structure has size $O(1)$. Then Step 2 can be done in time linear to the number of vertices checked for updating w . Note that w is updated to a point in $\partial P[w, u]$, and then the edges and vertices we check to update w change monotonically on ∂P in counterclockwise direction. Thus, the algorithm checks the edges and vertices of P for updating w in $O(n)$ time plus the time linear to the number of intervals in \mathcal{I}_L . Since there are $O(n + U)$ intervals in \mathcal{I}_L , the algorithm finds \bar{H}^* in $O(n + U)$ time using $O(n)$ space.

► **Lemma 10.** *We can compute the largest inscribed unit histogon of P that has three corner-contacts (type D) in $O(n + U)$ time using $O(n)$ space.*

We have shown how to find the largest inscribed unit histogons in P for types A, B, C and D. Taken together, we choose the largest one among them as the largest inscribed unit histogon in P . From Lemmas 3, 7, and 10, we have the following theorem.

► **Theorem 11.** *We can determine whether there is an inscribed unit histogon with positive height in P in $O(n)$ time. If exists, we can find a largest inscribed unit histogon in P in $O(n \log n + U)$ time using $O(n)$ space, where U is the number of intersections between the unit disks centered at the vertices of P and the edges of P .*

► **Remark.** One may wonder whether there are convex polygons with n vertices for which the number of v-intervals is $\Omega(n^2)$. We show how to construct such a convex polygon in Figure 4. The polygon has roughly $\lfloor \frac{n}{2} \rfloor$ vertices that are very close to each other and roughly $\lfloor \frac{n}{2} \rfloor$ edges each of which contains a point at distance 1 from those vertices as shown in the figure. Thus, there are $\Omega(n^2)$ feasible unit chords with different combinatorial structures.



■ **Figure 4** There can be $\Omega(n^2)$ v-event orientations.

4 Improved algorithms for a largest inscribed unit histogram

In this section, we assume that there exists a unit histogram of height δ inscribed in P for some $0 < \delta < 1$. Under this assumption, we show that a largest inscribed unit histogram in P can be computed in $O(n \log n + n/\delta)$ time, independent of the quantity U .

First, we show that there are $O(n)$ f-event orientations in total while θ increases from 0 to π , and that all f-event orientations can be computed in $O(n \log n)$ time.

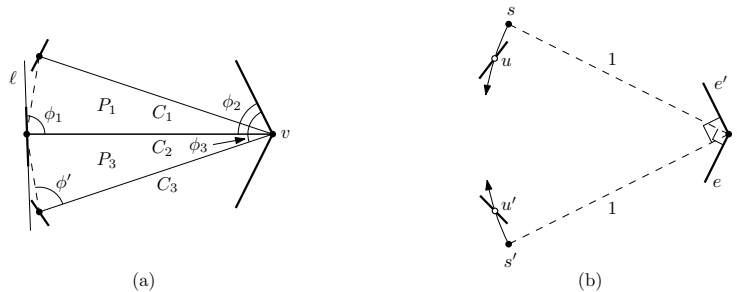
► **Lemma 12.** *There are $O(n)$ f-event orientations in total.*

Proof. There are at most two unit chords that are orthogonal to each edge of P . Thus there are $O(n)$ edge events.

Now we count the number of vertex events on a vertex v . Suppose that there are three vertex events on v , induced by unit chords C_1, C_2 and C_3 in increasing order of orientation. By definition, the orientations of C_1, C_2 and C_3 are all in O . Observe that both interior angles at v of the subpolygons of P induced by C_1 are acute. The same holds for C_2 and C_3 .

Let P_j be the subpolygon of P induced by C_2 that contains C_j for $j = 1, 3$. Let ℓ be the line containing the edge where the endpoint of C_2 , other than its endpoint lying on v , lies. See Figure 5(a). Then ℓ does not intersect the interior of C_3 . Since the isosceles triangle with two sides C_2 and C_3 has an acute angle at its base corners, the sum of the interior angles at the endpoints of C_2 in P_1 ($\phi_1 + \phi_2$ in Figure 5(a)) is strictly smaller than π . Similarly the sum of the interior angles at the endpoints of C_2 in P_3 is strictly smaller than π . Then C_2 is the longest chord contained in P at the orientation of C_2 . This contradicts that the orientation of C_2 is in O . Recall that an orientation θ is an element of O if the longest chord of orientation θ in P has length larger than 1. Thus there are at most two vertex events on a vertex and there are $O(n)$ vertex events in total. ◀

We show how to find all $O(n)$ f-events in $O(n \log n)$ time. The authors in the previous



■ **Figure 5** (a) $\phi_1' < \pi/2$ and $\phi_3 < \pi/2$ (acute). Since ℓ does not intersect the interior of C_3 , $\phi_1' + \phi_1 \leq \pi = \phi_3 - \phi_2 + 2\phi_1'$. $\phi_1 + \phi_2 \leq \phi_1' + \phi_3 < \pi$. (b) Circular arc queries for finding vertex event orientations.

paper [12] showed that two unit chords of a fixed orientation can be found in $O(\log n)$ time by binary search using the sorted list of vertices of P . For each edge of P , we can find all unit chords which are incident and orthogonal to the edge in $O(\log n)$ time. Thus we can find all edge events in $O(n \log n)$ time.

There are at most two vertex events on a vertex as shown in the proof of Lemma 12. To find them, we construct a data structure supporting circular ray shooting queries on P such that for a directed query arc of a circle with a start point and a direction (clockwise or counterclockwise), it finds the first intersection between the arc and ∂P . Cheng et al. [9] gave a hierarchical decomposition of a simple polygon for circular ray shooting queries of a fixed radius with $O(n \log n)$ construction time and $O(n)$ space that supports $O(\log n)$ query time. For a vertex v of P , let e and e' be the edges incident to v such that e' is the counterclockwise neighbor of v . For any line tangent to P at v , let s and s' be the points lying in the side of the tangent line containing P such that they are at distance 1 from v and the segments sv and $s'v$ are orthogonal to e and e' , respectively. See Figure 5(b) for an illustration. We perform two circular arc queries, one with the counterclockwise arc of the unit circle centered at v from s and the other with the clockwise arc of the unit circle centered at v from s' , in $O(\log n)$ time, and get the first intersections u and u' of the arcs with ∂P . Then we check whether a vertex event occurs at the orientations of two unit chords vu and vu' . Thus we can find all vertex events in P in $O(n \log n)$ time.

Let O_T be the set of all top feasible orientations, and O_B be the set of all bottom feasible orientations. Note that O_T and O_B induce intervals contained in O and their endpoints are f -events by definition. Let $\mathcal{I}(O_T)$ and $\mathcal{I}(O_B)$ be the set of intervals induced by O_T and O_B , respectively. By Lemma 12, there are $O(n)$ intervals in $\mathcal{I}(O_T)$ and $\mathcal{I}(O_B)$. For a top feasible orientation $\theta \in O_T$, let $\bar{H}(\theta)$ be the largest unit histogram inscribed in P with top side lying on the top unit chord of orientation θ . Let $I \in \mathcal{I}(O_T)$ be an interval with endpoints θ_0, θ_1 satisfying $\theta_0 < \theta_1$. For any $\theta \in I$, the top side $\alpha(\theta) = \alpha_1(\theta)\alpha_2(\theta)$ of $\bar{H}(\theta)$ is a unit chord, and moves continuously as θ continuously increases in I by Lemma 6. Hence, its limits at the endpoints θ_0 and θ_1 are well defined as $\alpha(\theta_0)$ and $\alpha(\theta_1)$, so we can discuss the top side $\alpha(\theta)$ over all $\theta \in \text{cl}(I)$ in the closure of each interval $I \in \mathcal{I}(O_T)$, including its endpoints.

It is possible that two consecutive intervals $I, I' \in \mathcal{I}(O_T)$ share an endpoint θ' , while θ' belongs to none of I and I' . In this case, the limits of $\alpha(\theta)$ at θ' over I and I' may not agree. In the following, we handle each interval of $\mathcal{I}(O_T)$ separately in order, so we abuse a notation to mean $\alpha(\theta')$ by the limit over the interval we are currently handling. This will be clear from context.

4.1 Orientations dominated by other orientations

Let $\mathcal{C}(\theta)$ denote $\partial P[\alpha_1(\theta), \alpha_2(\theta)]$ for an orientation $\theta \in \text{cl}(I)$ with $I \in \mathcal{I}(O_T)$. Since the interior angles at the endpoints of $\alpha(\theta)$ in $P[\alpha_2(\theta), \alpha_1(\theta)]$ are at least $\pi/2$, the endpoints of $\alpha(\theta)$ move continuously along ∂P in counterclockwise direction as θ continuously increases in I . This implies that $\mathcal{C}(\theta) \not\subseteq \mathcal{C}(\theta')$ and $\mathcal{C}(\theta') \not\subseteq \mathcal{C}(\theta)$ for any $\theta, \theta' \in \text{cl}(I)$ with $\theta \neq \theta'$.

► **Observation 13.** For $I \in \mathcal{I}(O_T)$ and $\theta \neq \theta' \in \text{cl}(I)$, we have $\mathcal{C}(\theta) \not\subseteq \mathcal{C}(\theta')$ and $\mathcal{C}(\theta') \not\subseteq \mathcal{C}(\theta)$.

► **Lemma 14.** For $I, I' \in \mathcal{I}(O_T)$ with $I \neq I'$, $\theta \in \text{cl}(I)$, and $\theta' \in \text{cl}(I')$, it holds that $|\bar{H}(\theta)| < |\theta - \theta'|$ if $\mathcal{C}(\theta) \subseteq \mathcal{C}(\theta')$.

For $I, I' \in \mathcal{I}(O_T)$ with $I \neq I'$, $\theta \in \text{cl}(I)$, and $\theta' \in \text{cl}(I')$, we say θ is δ -dominated by θ' (or θ' δ -dominates θ) if $\mathcal{C}(\theta) \subseteq \mathcal{C}(\theta')$ and $|\theta - \theta'| \leq \delta$. By Observation 13, θ and θ' are contained in two distinct intervals of $\mathcal{I}(O_T)$ if θ δ -dominates θ' or θ' δ -dominates θ . If θ is δ -dominated

13:12 Inscribing or Circumscribing a Histogram to a Convex Polygon

by θ' , we have $|\bar{H}(\theta)| < \delta$ by Lemma 14. Observe, however, that there can be orientations $\theta'' \in O_T$ with $|\bar{H}(\theta'')| < \delta$ that are not dominated by any other orientations contained in an interval of $\mathcal{I}(O_T)$. We use the following lemma to determine whether an orientation $\theta \in I$ is δ -dominated by another orientation $\theta' \in I'$ for $I, I' \in \mathcal{I}(O_T)$.

► **Lemma 15.** *Let $\theta_1, \theta_2, \theta$ be three orientations, each contained in the closure of an interval in $\mathcal{I}(O_T)$, with $\theta_1 < \theta < \theta_2$ or $\theta_2 < \theta < \theta_1$. Suppose that $\mathcal{C}(\theta_1) \not\subseteq \mathcal{C}(\theta)$ and $\mathcal{C}(\theta) \not\subseteq \mathcal{C}(\theta_1)$. Then, we have $\mathcal{C}(\theta) \subseteq \mathcal{C}(\theta_2)$ if $\mathcal{C}(\theta_1) \subseteq \mathcal{C}(\theta_2)$, and $\mathcal{C}(\theta_2) \subseteq \mathcal{C}(\theta)$ if $\mathcal{C}(\theta_2) \subseteq \mathcal{C}(\theta_1)$.*

Lemma 15 provides us a tool to infer the δ -dominance relation over orientations in each interval of O_T , namely, if θ_1 is δ -dominated by θ_2 , then θ is also δ -dominated by θ_2 , and if θ_2 is δ -dominated by θ_1 , then θ_2 is also δ -dominated by θ . We use this, together with Observation 13 and Lemma 14, to remove as much δ -dominated orientations as possible from each interval in $\mathcal{I}(O_T)$, resulting in relevant subintervals.

Removing δ -dominated orientations. Let $I, I' \in \mathcal{I}(O_T)$ be two distinct intervals with $\text{cl}(I) = [\theta_0, \theta_1]$ and $\text{cl}(I') = [\theta'_0, \theta'_1]$ such that an orientation $\theta \in \text{cl}(I)$ is δ -dominated by an orientation $\theta' \in \text{cl}(I')$. By Lemma 15, every orientation in $[\theta, \theta_1]$ is δ -dominated by θ'_0 if $\theta < \theta'$, and every orientation in $[\theta_0, \theta]$ is δ -dominated by θ'_1 if $\theta > \theta'$. Thus, we can determine whether an orientation is δ -dominated by another orientation using the endpoints of intervals in $\mathcal{I}(O_T)$. Moreover, the set of orientations in I which are not δ -dominated by other orientations appears as a subinterval of I unless it is empty.

In the following, we describe how to remove δ -dominated orientations from the intervals of $\mathcal{I}(O_T)$. Note that this procedure does not remove all δ -dominated orientations but the δ -dominated orientations that remain after the procedure have some property as shown in Lemma 16. We process the intervals of $\mathcal{I}(O_T)$ one by one in increasing order of orientation. In the course, we maintain a sequence L of (sub)intervals that have been processed so far after removing the orientations δ -dominated by some other orientations in the closure of an interval of $\mathcal{I}(O_T)$. Initially, L is set to an empty list.

Imagine that we have processed the first i intervals of $\mathcal{I}(O_T)$ and we are about to process the $(i+1)$ -th interval I of $\mathcal{I}(O_T)$ with $\text{cl}(I) = [\theta_0, \theta_1]$. Let $L = \langle [\theta_0^1, \theta_1^1], [\theta_0^2, \theta_1^2], \dots, [\theta_0^m, \theta_1^m] \rangle$ be the sequence of intervals that have been processed so far. If L is empty, we simply append $\text{cl}(I)$ into L . Otherwise, we update L by removing (sub)intervals of L or a (sub)interval of $\text{cl}(I)$ consisting of the orientations δ -dominated by other orientations by the following rules. If $\theta_0 = \theta_1^m$, we set θ_1^m to be $\theta_1^m - \varepsilon$ for infinitesimally small $\varepsilon > 0$ temporarily whenever we check the δ -dominance between θ_0 and θ_1^m .

Rule 1. If θ_0 and θ_1^m do not δ -dominate each other, append $\text{cl}(I) = [\theta_0, \theta_1]$ to L .

Rule 2. If θ_1^m is δ -dominated by θ_0 ,

- a. find the smallest integer $0 < j \leq m$ such that θ_1^j is δ -dominated by θ_0 ,
- b. find the smallest orientation $r \in [\theta_0^j, \theta_1^j]$ such that r is δ -dominated by θ_0 ,
- c. remove $[\theta_0^j, \theta_1^j], [\theta_0^{j+1}, \theta_1^{j+1}], \dots, [\theta_0^m, \theta_1^m]$ from L , and
- d. append $[\theta_0^j, r]$ to L if $\theta_0^j < r$, and then append $\text{cl}(I) = [\theta_0, \theta_1]$ to L .

Rule 3. If θ_0 is δ -dominated by θ_1^m ,

- a. find the largest orientation $r \in [\theta_0, \theta_1]$ such that r is δ -dominated by θ_1^m , and
- b. append $[r, \theta_1]$ to L if $r < \theta_1$.

After removing δ -dominated orientations from the intervals of $\mathcal{I}(O_T)$ by the procedure above, the list L consists of $O(n)$ intervals and has the following property.

► **Lemma 16.** *For any two orientations θ, θ' contained in some intervals of L , $|\theta - \theta'| = \delta$ if θ is δ -dominated by θ' .*

Analysis. We maintain the list L of intervals in increasing order of orientation. Let $I \in \mathcal{I}(O_T)$ with $\text{cl}(I) = [\theta_0, \theta_1]$ be the interval that we are about to process for $L = \langle [\theta_0^1, \theta_1^1], [\theta_0^2, \theta_1^2], \dots, [\theta_0^m, \theta_1^m] \rangle$. The last interval $[\theta_0^m, \theta_1^m]$ of L can be found in $O(1)$ time. We can check if Rule 1 applies and append $[\theta_0, \theta_1]$ to L in $O(1)$ time. When Rule 2 applies, we find all intervals $[\theta'_0, \theta'_1]$ in L such that θ'_1 is δ -dominated by θ_0 by checking the intervals in L one by one in decreasing order of orientation. By Lemma 15, those intervals form a contiguous subsequence $\langle [\theta_0^j, \theta_1^j], [\theta_0^{j+1}, \theta_1^{j+1}], \dots, [\theta_0^m, \theta_1^m] \rangle$ of L . We remove them from L in time linear to the number of removed intervals. Then we find the smallest orientation r in $[\theta_0^j, \theta_1^j]$ such that r is δ -dominated by θ_0 . If $\theta_0^j < r$, then either it holds that $|\theta_0 - r| = \delta$ or $\mathcal{C}(\theta_0)$ and $\mathcal{C}(r)$ share the common endpoint $\alpha_1(\theta_0) = \alpha_1(r)$. In the former case, we find $r \in [\theta_0^j, \theta_1^j]$ such that $|\theta_0 - r| = \delta$ in $O(1)$ time. In the latter case, we find $\alpha_2(r)$ using a circular arc query in $O(\log n)$ time with the directed arc of unit circle centered at $\alpha_1(\theta_0)$, since there is at most one orientation θ in $[\theta_0^j, \theta_1^j]$ such that $\alpha_1(\theta) = \alpha_1(\theta_0)$. When Rule 3 applies, we find the largest orientation $r \in [\theta_0, \theta_1]$ such that r is δ -dominated by θ_1^m for the last interval $[\theta_0^m, \theta_1^m]$ in L using a circular arc query in $O(\log n)$ time. Since there are $O(n)$ intervals in $\mathcal{I}(O_T)$, we can process them in $O(n \log n)$ time using $O(n)$ space.

4.2 Orientations θ with $|\bar{H}(\theta)| = \delta$.

Let I be an interval in L . Observe that the height of $\bar{H}(\theta)$ is positive and changes continuously as θ continuously increases in I since the endpoints of $\alpha(\theta)$ and the two chords orthogonal to $\alpha(\theta)$ through $\alpha_1(\theta)$ and $\alpha_2(\theta)$ move continuously along ∂P by Lemma 6. We call each orientation $\theta \in I$ such that $|\bar{H}(\theta)| = \delta$ a δ -event orientation. The set of δ -event orientations in I partitions I into subintervals I' such that either $|\bar{H}(\theta)| < \delta$ for all $\theta \in \text{int}(I')$ or $|\bar{H}(\theta)| > \delta$ for all $\theta \in \text{int}(I')$.

Finding δ -event orientations. We find all δ -event orientations contained in intervals of L as follows. A rectangle R of an orientation $\theta \in [0, \pi)$ is a copy of an axis-aligned rectangle \bar{R} obtained by rotating \bar{R} by θ in counterclockwise direction. The width of R is the width of \bar{R} , and the top and bottom sides of R are the ones corresponding to the top and bottom sides of \bar{R} , respectively.

First, we compute two lists L_T and L_B of intervals in addition to L . We compute the set O_T^δ of orientations θ such that the largest inscribed δ -width rectangle of orientation θ with top side lying on a chord of length δ has a positive height. Then we remove orientations $\theta \in I$ such that $|\theta - \theta'| \leq 1/\delta$ and $\partial P[\gamma_1(\theta), \gamma_2(\theta)] \subseteq \partial P[\gamma_1(\theta'), \gamma_2(\theta')]$ for some other orientation $\theta' \in I'$ for $I, I' \in \mathcal{I}(O_T^\delta)$, where $\gamma_1(\theta)\gamma_2(\theta)$ is the top chord of length δ and orientation θ in P . This can be done by using the same procedure for computing L . Then we obtain L_T from the intervals of $\mathcal{I}(O_T^\delta)$. Similarly, we compute L_B from the set of orientations θ such that the largest inscribed δ -width rectangle of orientation θ with bottom side lying on a chord of length δ has a positive height.

If $|\bar{H}(\theta)| = \delta$ and $\bar{H}(\theta)$ has the bottom-left corner lying on ∂P for some $\theta \in O_T$, the largest inscribed δ -width rectangle of orientation $\theta + \pi/2$ with top side lying on a chord of length δ has height 1, and thus $\theta + \pi/2$ is contained in an interval of L_T . Similarly, $\theta - \pi/2$ is contained in an interval of L_B if $|\bar{H}(\theta)| = \delta$ and $\bar{H}(\theta)$ has the bottom-right corner lying on ∂P with height δ . Therefore, to compute δ -event orientations, we process the intervals of L, L_T and L_B in increasing order of orientation. For an interval $I \in L$, we can compute all v -event orientations in I corresponding to top unit chords and the subintervals into which I is partitioned by the v -event orientations. We can also compute the subintervals of L_T and

13:14 Inscribing or Circumscribing a Histogram to a Convex Polygon

L_B using ν -event orientations corresponding to top chords of length δ . Those subintervals are computed in increasing order of orientation and the combinatorial structure of the top chord (of length 1 or δ) of orientations θ is invariant for any θ contained in an interval.

Whenever a subinterval $I = [\theta_0, \theta_1]$ of L is identified by a ν -event orientation, we check if there is an orientation $\theta \in I$ such that $|\bar{H}(\theta)| = \delta$. Let I' be the subinterval of L_T containing $\theta_0 + \pi/2$, among those partitioned by the ν -event orientations for the top chords of length δ . Using the combinatorial structures of the top unit chord of orientation $\theta \in I$ and the top chord of length δ in I' , we can build an equation that has a solution $\theta' \in I$ if and only if $|\bar{H}(\theta')| = \delta$. By solving the equation, we compute all orientations θ in I such that $|\bar{H}(\theta)| = \delta$ and $\bar{H}(\theta)$ has the bottom-left corner lying on ∂P . We also compute all δ -event orientations θ in I such that $\bar{H}(\theta)$ has the bottom-right corner lying on ∂P using $I = [\theta_0, \theta_1]$ and the subinterval of L_B containing $\theta_0 - \pi/2$. Similarly, whenever a subinterval of $L_{\pi/2}$ or $L_{-\pi/2}$ is identified by a ν -event orientation, we compute all δ -event orientations in the subinterval. After processing all subintervals of L , L_T and L_B , we obtain all δ -event orientations.

δ -feasible subintervals. We partition the intervals of $\mathcal{I}(O_T)$ into subintervals by δ -event orientations. Then we can obtain the set of subintervals I' such that θ is a top feasible orientation and $|\bar{H}(\theta)| \geq \delta$ for all $\theta \in I'$. Similarly, we obtain the set of subintervals I'' from $\mathcal{I}(O_B)$ such that θ is a bottom feasible orientation and $|\bar{H}(\theta)| \geq \delta$ for all $\theta \in I''$. We apply the algorithms for computing the largest inscribed unit histograms in P of types C (Section 3.2) and D (Section 3.3) with the subintervals of $\mathcal{I}(O_T)$ and $\mathcal{I}(O_B)$ induced by δ -event orientations, instead of the intervals of $\mathcal{I}(O)$.

Analysis. Consider two orientations θ_1 and θ_2 , each contained in an interval of L , such that both $\alpha_1(\theta_1)$ of $\alpha(\theta_1)$ and $\alpha_1(\theta_2)$ of $\alpha(\theta_2)$ lie on the same vertex of P . By Lemma 16, $|\theta_1 - \theta_2| \geq \delta$ since $\mathcal{C}(\theta_1) \subseteq \mathcal{C}(\theta_2)$ or $\mathcal{C}(\theta_2) \subseteq \mathcal{C}(\theta_1)$. Thus, for each vertex of P , there are $O(1/\delta)$ orientations in the intervals of L at which the top unit chord has its endpoint at the vertex. So there are $O(n/\delta)$ ν -event orientations in the intervals of L . Similarly, we can show that the number of ν -event orientations in the intervals of L_T and L_B is $O(\delta n) = O(n)$ since $\delta < 1$. Then, the total number of subintervals of L , L_T and L_B induced by ν -event orientations is $O(n/\delta)$. Given L , L_T and L_B , we can compute the subintervals in $O(n/\delta)$ time while tracing the top chords (of length 1 or δ) in P .

Recall that the combinatorial structure of chords (of length 1 or δ) of orientations in the interior of a subinterval remains the same. Then the equation for computing δ -event orientations is of the form $\sin(\theta + A) + B = \delta \sin(\theta + C)$ using the law of sines, where A , B and C are all constants. Thus, the equation can be solved in $O(1)$ time, resulting in $O(1)$ solutions that correspond to δ -event orientations. Since there are $O(n/\delta)$ pairs of subintervals of L and L_T (and L and L_B) which overlap each other, there are $O(n/\delta)$ δ -event orientations in the intervals of $\mathcal{I}(O_T)$ and they can be computed in $O(n/\delta)$ time.

We compute L , L_T and L_B in $O(n \log n)$ time and the δ -event orientations in $O(n/\delta)$ time using $O(n)$ space. Moreover, the subintervals of $\mathcal{I}(O_T)$ induced by δ -event orientations have $O(n/\delta)$ ν -event orientations in total. Thus, the running time of the algorithms for computing the largest histogram of types C and D in P decreases to $O(n \log n + n/\delta)$ if we use the subintervals of $\mathcal{I}(O_T)$ and $\mathcal{I}(O_B)$ induced by δ -event orientations, instead of the intervals of $\mathcal{I}(O)$.

► **Theorem 17.** *Suppose there is a unit histogram of height δ inscribed in P for $\delta < 1$. A largest inscribed unit histogram in P can be computed in $O(n \log n + n/\delta)$ time and $O(n)$ space.*

Our algorithm works under the assumption that there exists a unit histogon contained in P whose height is at least δ , while it can be used to test the existence for any $0 < \delta < 1$. Given δ as input, if there exists such a histogon, then Theorem 17 applies and our algorithm returns a largest inscribed unit histogon in P . Otherwise, if not, all orientations in O_T and O_B are removed by the δ -dominance relation or there is no δ -event orientation. Hence, this case can be identified when the list L is turned to be empty or when no δ -event orientation is identified. In the former case, it is obvious that there is no unit histogon of height at least δ . In the latter case, the height of the unit histogon $\bar{H}(\theta)$ is either larger than δ for all $\theta \in O_T$ or smaller than δ for all $\theta \in O_T$. Thus, by picking one orientation θ from O_T and computing $\bar{H}(\theta)$, one can check which case this is.

► **Corollary 18.** *Given $0 < \delta < 1$, one can determine whether there exists an inscribed unit histogon with height δ in P in $O(n \log n + n/\delta)$ time using $O(n)$ space.*

Note that we can compute a largest inscribed unit histogon in $O(n \log n)$ time if its height h is $\Omega(1/\log n)$ by setting $\delta = 1/\log n$ in Corollary 18 and Theorem 17. Otherwise, we can get an output-sensitive algorithm for finding a largest inscribed unit histogon as follows. We search a value h_0 with $0 < h_0 \leq h$ for the height h of a largest inscribed unit histogon using Corollary 18 with a sequence of δ values, $\delta_i = 2^{-2^i}$ for $i = 0, 1, \dots, \lceil \log \log(1/h) \rceil$. Then we apply Theorem 17 with h_0 . Observe that $2^{-2^m} \leq h < 2^{-2^{m-1}}$ for some nonnegative integer m . Then $m < \log \log(1/h) + 1$ and $2^{-2^m} > h^2$. This results in running time $O(n \log n \log \log(1/h) + n/h^2)$. We can even improve it by starting with the test value δ_j such that $\delta_j < 1/\log n \leq \delta_{j-1}$, concluding the following.

► **Corollary 19.** *Let h be the height of a largest inscribed unit histogon in P . A largest inscribed unit histogon in P can be computed in $O(n \log n)$ time if $h = \Omega(\frac{1}{\log n})$, or in $O(n \log n \log \log \frac{1}{h \log n} + \frac{n}{h^2})$ time, otherwise.*

5 Largest inscribed histogon and smallest circumscribed histogon

We also considered three optimization problems for a convex polygon P with n vertices. The term $T(m)$ appearing in the running times in the following denotes the time complexity of the optimization step for the trigonometric expression with $O(m)$ terms, each of quadratic form. Barequet and Rogol [4] observed that $T(m) = O(m)$ in practice.

- **Largest inscribed k -histogon.** Given a positive integer $k > 1$, find a largest k -histogon of arbitrary orientation inscribed in P . We present an algorithm for this problem that runs in $O(kn^2(\log n + kT(\min\{k, n\})))$ time using $O(\min\{k, n\}n)$ space.
- **Largest inscribed histogon.** Find the largest histogon of arbitrary orientation inscribed in P . We present an algorithm for this problem that runs in $O(Dn^2(\log n + T(\min\{D, n\})))$ time using $O(\min\{D, n\}n)$ space, where D denotes the diameter of P .
- **Smallest circumscribed histogon.** Find the smallest histogon of arbitrary orientation circumscribed to P . We present an algorithm for this problem that runs in $O(Dn(\log(\min\{D, n\}) + T(\min\{D, n\})))$ time using $O(n)$ space, where D denotes the diameter of P .

References

- 1 Alok Aggarwal, Maria M Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.
- 2 Hee-Kap Ahn, Sang Won Bae, Otfried Cheong, and Joachim Gudmundsson. Aperture-angle and hausdorff-approximation of convex figures. *Discrete & Computational Geometry*, 40(3):414–429, 2008.

13:16 Inscribing or Circumscribing a Histogram to a Convex Polygon

- 3 Hee-Kap Ahn, Peter Brass, Otfried Cheong, Hyeon-Suk Na, Chan-Su Shin, and Antoine Vigneron. Inscribing an axially symmetric polygon and other approximation algorithms for planar convex sets. *Computational Geometry*, 33(3):152–164, 2006.
- 4 Gill Barequet and Vadim Rogol. Maximizing the area of an axially symmetric polygon inscribed in a simple polygon. *Computers & Graphics*, 31(1):127–136, 2007.
- 5 Binay Bhattacharya and Asish Mukhopadhyay. On the minimum perimeter triangle enclosing a convex polygon. In *Proc. Japanese Conference on Discrete and Computational Geometry (JCDCG 2002)*, pages 84–96, 2002.
- 6 Sergio Cabello, Otfried Cheong, Christian Knauer, and Lena Schlipf. Finding largest rectangles in convex polygons. *Computational Geometry*, 51:67–74, 2016.
- 7 Tauã M. Cabreira, Lisane B. Brisolará, and Paulo R. Ferreira Jr. Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1), 2019.
- 8 Jyun-Sheng Chang and Chee-Keng Yap. A polynomial solution for the potato-peeling problem. *Discrete & Computational Geometry*, 1(2):155–182, 1986.
- 9 Siu-Wing Cheng, Otfried Cheong, Hazel Everett, and Rene Van Oostrum. Hierarchical decompositions and circular ray shooting in simple polygons. *Discrete & Computational Geometry*, 32(3):401–415, 2004.
- 10 Yujin Choi, Seungjun Lee, and Hee-Kap Ahn. Maximum-area and maximum-perimeter rectangles in polygons. *Computational Geometry*, 94:101710, 2021.
- 11 Howie Choset. Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, October 2001.
- 12 Jaehoon Chung, Sang Won Bae, Chan-Su Shin, Sang Duck Yoon, and Hee-Kap Ahn. Approximating convex polygons by histograms. In *Proc. 34th Canadian Conference on Computational Geometry (CCCG 2022)*, 2022.
- 13 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*, chapter Point Location, pages 121–146. Springer-Verlag TELOS, 3rd edition, 2008.
- 14 A DePano, Yan Ke, and Joseph O’Rourke. Finding largest inscribed equilateral triangles and squares. In *Proc. 25th Allerton Conference on Communications, Control and Computing*, pages 869–878, 1987.
- 15 Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.
- 16 Martin Held. *On the Computational Geometry of Pocket Machining*. Lecture Notes in Computer Science. Springer Verlag, 1991.
- 17 Kai Jin and Kevin Matulef. Finding the maximum area parallelogram in a convex polygon. In *Proc. 23rd Canadian Conference on Computational Geometry (CCCG 2011)*, 2011.
- 18 Christian Knauer, Lena Schlipf, Jens M Schmidt, and Hans Raj Tiwary. Largest inscribed rectangles in convex polygons. *Journal of discrete algorithms*, 13:78–85, 2012.
- 19 Seungjun Lee, Taekang Eom, and Hee-Kap Ahn. Largest triangles in a polygon. *Computational Geometry*, 98:101792, 2021.
- 20 Eleftherios A. Melissaratos and Diane L. Souvaine. On solving geometric optimization problems using shortest paths. In *Proc. 6th Annual Symposium on Computational Geometry (SCG ’90)*, pages 350–359, 1990.
- 21 Joseph O’Rourke, Alok Aggarwal, Sanjeev Maddila, and Michael Baldwin. An optimal algorithm for finding minimal enclosing triangles. *Journal of Algorithms*, 7(2):258–269, 1986.
- 22 Christian Schwarz, Jürgen Teich, Alek Vainshtein, Emo Welzl, and Brian L. Evans. Minimal enclosing parallelogram with application. In *Proc. 11th Annual Symposium on Computational Geometry (SCG ’95)*, pages 434–435, 1995.
- 23 Godfried Toussaint. Solving geometric problems with the rotating calipers. In *Proc. Mediterranean Electrotechnical Conference (MELECON’83)*, 1983.
- 24 Ivor van der Hoog, Vahideh Keikha, Maarten Löffler, Ali Mohades, and Jérôme Urhausen. Maximum-area triangle in a convex polygon, revisited. *Information Processing Letters*, 161:105943, 2020.

More Verifier Efficient Interactive Protocols for Bounded Space

Joshua Cook   

University of Texas at Austin, TX, USA

Abstract

Let $\mathbf{TISP}[T, S]$, $\mathbf{BPTISP}[T, S]$, $\mathbf{NTISP}[T, S]$ and $\mathbf{CoNTISP}[T, S]$ be the set of languages recognized by deterministic, randomized, nondeterministic, and co-nondeterministic algorithms, respectively, running in time T and space S . Let $\mathbf{ITIME}[T_V, T_P]$ be the set of languages recognized by an interactive protocol where the verifier runs in time T_V and the prover runs in time T_P .

For $S = \Omega(\log(n))$ and T constructible in time $\log(T)S + n$, we prove:

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}] \quad (1)$$

$$\mathbf{BPTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}] \quad (2)$$

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)^2 S + n), 2^{O(S)}] \quad (3)$$

$$\mathbf{CoNTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)^2 S + n), 2^{O(S)}]. \quad (4)$$

The best prior verifier time is from Shamir [21, 11]:

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)(S + n)), 2^{O(\log(T)(S+n))}].$$

Our prover is faster, and our verifier is faster when $S = o(n)$.

The best prior prover time uses ideas from Goldwasser, Kalai, and Rothblum [9]:

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S^2 + n), 2^{O(S)}].$$

Our verifier is faster when $\log(T) = o(S)$, and for deterministic algorithms.

To our knowledge, no previous interactive protocol for \mathbf{TISP} simultaneously has the same verifier time and prover time as ours. In our opinion, our protocol is also simpler than previous protocols.

2012 ACM Subject Classification Theory of computation \rightarrow Interactive proof systems

Keywords and phrases Interactive Proofs, Verifier Time, Randomized Space, Nondeterministic Space, Fine Grain Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.14

Related Version *Full Version:* <https://ecc.weizmann.ac.il/report/2022/093/>

Funding Funded by NSF grant number 1705028 and 2200956.

Acknowledgements Thanks to Dana Moshkovitz for suggestions on writing and presentation, and Tayvin Otti for spellchecking.

1 Introduction

One of the most celebrated results of computer science is the proof that $\mathbf{IP} = \mathbf{PSPACE}$ [21, 11]. Any language computable in polynomial space can be verified in polynomial time by a verifier with access to randomness and an untrusted, computationally unbounded prover.

Interactive proofs have many applications, for example proving circuit lower bounds for $\mathbf{MA}/1$ [19], and for \mathbf{NQP} [14]. More verifier time efficient \mathbf{PCPs} [13] improve the results of [19]. Even pseudo random generators [5] use interactive proofs [9].



© Joshua Cook;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 14; pp. 14:1–14:18



Leibniz International Proceedings in Informatics

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

14:2 More Verifier Efficient Interactive Protocols for Bounded Space

The previous best verifier time in an interactive protocol for an algorithm running in time T and space S was by Shamir [21], whose verifier runs in time $\tilde{O}(\log(T)S)$ for $S = \Omega(n)$. We improve this result to apply to any $\log(T)S = \Omega(n)$. Our prover is also more efficient. For instance, if $S = \sqrt{n}$ and $T = 2^{\sqrt{n}}$, then we show that

$$\mathbf{TISP}[2^{\sqrt{n}}, \sqrt{n}] \subseteq \mathbf{ITIME}[\tilde{O}(n), 2^{O(\sqrt{n})}],$$

while Shamir only gives

$$\mathbf{TISP}[2^{\sqrt{n}}, \sqrt{n}] \subseteq \mathbf{ITIME}[\tilde{O}(n^{1.5}), 2^{O(n^{1.5})}].$$

That is, our verifier only requires time $\tilde{O}(n)$, but Shamir's requires time $\tilde{O}(n^{1.5})$. Our prover only requires time $2^{O(\sqrt{n})}$, but Shamir's requires time $2^{O(n^{1.5})}$.

The previous best prover time in an interactive protocol for an algorithm running in time T and space S was by Goldwasser, Kalai and Rothblum [9], whose prover runs in a similar time to ours, but whose verifier requires time $\log(T)S^2$. We improve the verifier time by making the quadratic dependence on S linear. If $T = \mathbf{poly}(S)$, our protocol improves the verifier time from $\tilde{O}(S^2)$ to $\tilde{O}(S)$. If $T = 2^{O(S)}$, our protocol improves the verifier time from $\tilde{O}(S^3)$ to $\tilde{O}(S^2)$.

Our results prove a more direct, more efficient, and (in our opinion) simpler protocol than that given in [21] using sum check [11]. We use a reduction to matrix exponentiation instead of quantified Boolean formulas, and a direct arithmetization of the algorithm instead of a Boolean formula. We then apply this protocol to the special cases of deterministic, randomized, and nondeterministic algorithms.

1.1 Results

Let $\mathbf{ITIME}[T_V, T_P]$ be the class of languages computed by an interactive protocol where the verifier runs in time T_V and the prover runs in time T_P . Similarly, let $\mathbf{ITIME}^1[T_V, T_P]$ be the same with perfect completeness. Let $\mathbf{TISP}[T, S]$ be the class of languages computable in simultaneous time T and space S . Our first result is:

► **Theorem 1** (Efficient Interactive Protocol For **TISP**). *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

Our protocol has several other desirable properties. The verifier only needs space $\tilde{O}(S)$. This protocol is also public coin, non adaptive, and unambiguous (as described in [16]). This protocol can also verify an $O(\log(T)S + n)$ bit output, not just membership in a language.

We note that $L \in \mathbf{ITIME}^1[T_V, T_P]$ implies that $L \in \mathbf{SPACE}[O(T_V)]$, since a prover can find an optimal prover strategy in a space efficient way. Thus our dependence on S is essentially optimal. It is open whether one can remove the $\log(T)$ factor.

Using Nisan's PRG for bounded space [15], we extend Theorem 1 to get a similar result for randomized bounded space algorithms. Let $\mathbf{BPTISP}[T, S]$ be the class of languages computable in simultaneous randomized time T and space S .

► **Theorem 2** (Efficient Interactive Protocol For **BPTISP**). *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{BPTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

If one tries to use Nisan's PRG with Shamir's protocol, you increase the input size to $\log(T)S$, which gives a time $\tilde{O}(\log(T)^2S + \log(T)n)$ verifier. One can also use Saks and Zhou [18] to reduce $\mathbf{BSPACE}[S]$ to $\mathbf{SPACE}[S^{1.5}]$, then apply an \mathbf{IP} , but this also only gives a time S^3 verifier.

The protocol used for Theorem 1 internally counts the number of accepting paths in a nondeterministic algorithm, modulo a prime. By appropriately sampling a prime, we can get an efficient interactive protocol for \mathbf{NTISP} .

► **Theorem 3** (Efficient Interactive Protocol For \mathbf{NTISP}). *Let S and T be computable in time $\tilde{O}(\log(T)^2S + n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{NTISP}[T, S] \cup \mathbf{CoNTISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)^2S + n), 2^{O(S)}].$$

While $\mathbf{NSPACE}[O(S)] = \mathbf{Co-NSPACE}[O(S)]$ [10, 23], the same result is not known for time bounded computation. So the case for \mathbf{NTISP} and $\mathbf{CoNTISP}$ are both interesting and potentially different.

This version of the paper only proves the protocol for deterministic space. To see the rest of the proofs, see the full paper [6].

1.2 Related Work

This work builds on techniques used by Lund, Fortnow, Karloff and Nisan [11] to prove that $\#\mathbf{P} \in \mathbf{IP}$ and extended by Shamir [21] to show that $\mathbf{PSPACE} = \mathbf{IP}$. Shamir used Savitch's theorem [20] to reduce space bounded computation to a quantified Boolean formula, and then gave a sum check similar to [11] to verify it.

Although it was not shown, the bounded space variation of Shamir's protocol (given at the end of [21]) can be implemented time efficiently for the verifier.¹

► **Theorem 4** (Shamir's Protocol). *Let S and T be time $\tilde{O}(\log(T)(S + n))$ computable with $S = \Omega(\log(n))$. Then*

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)(S + n)), 2^{O(\log(T)(S+n))}].$$

One can extend Shamir's protocol to nondeterministic algorithms using other ideas in the same paper to get

► **Theorem 5** (Shamir's Protocol for Nondeterministic Algorithms). *Let S and T be time $\tilde{O}(\log(T)^2(S + n))$ computable with $S = \Omega(\log(n))$. Then*

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)^2(S + n)), 2^{O(\log(T)(S+n))}].$$

It is not clear from Shamir's work how to get an efficient prover, or how to handle the case where $S = o(n)$.

Shen [22] gave a highly influential variation of Shamir's proof that is frequently taught (for instance [1]). Shen's result gives a less efficient verifier. Meir gave a proof that $\mathbf{IP} = \mathbf{PSPACE}$ [12] which uses a different kind of sum check with a different kind of code. Or Meir's approach also does not improve the verifier time, but introduces useful techniques [17].

¹ Shamir's reduction from general quantified Boolean formulas is not this efficient. The low space or time for the verifier uses the specific form given by the reduction from bounded space to a quantified Boolean formula.

14:4 More Verifier Efficient Interactive Protocols for Bounded Space

Sum check was also used by Babai, Fortnow and Lund [4] to prove that $\text{MIP} = \text{NEXP}$. This line of work is foundational to many **PCPs** [3, 2].

In a very influential paper, Goldwasser, Kalai and Rothblum gave doubly efficient interactive proofs for depth bounded computation [9]. Doubly efficient proofs are proofs where the prover runs in time polynomial in the algorithm it wishes to prove. GKR is very efficient for uniform, low depth algorithms, like those in **NC**, both for the verifier and the prover.

► **Theorem 6 (GKR For Depth)**. *Let L be a language computed by a family of $O(\log(w))$ -space uniform Boolean circuits of width w and depth d where w and d are computable in time $(n + d)\text{polylog}(w)$. Then*

$$L \in \text{ITIME}^1[(n + d)\text{polylog}(w), \text{poly}(wd)].$$

A space S and a time T nondeterministic algorithm, A , can be converted to a width $2^{O(S)}$ and depth $O(\log(T)S)$ circuit, C , using repeated squaring on the adjacency matrix of A 's computation graph. Using Theorem 6 with this circuit we get a protocol for bounded space. The circuit is very simple, so we believe² the $\text{polylog}(w) = \text{poly}(S)$ term can be made $\tilde{O}(S)$. This gives:

► **Theorem 7 (GKR for NSPACE)**. *Let S and T be time $\tilde{O}(\log(T)S^2 + n)$ computable with $S = \Omega(\log(n))$. Then*

$$\text{NTISP}[T, S] \cup \text{CoNTISP}[T, S] \subseteq \text{ITIME}^1[\tilde{O}(\log(T)S^2 + n), 2^{O(S)}].$$

The GKR protocol, as well as ours, only have polynomial time provers when $T = 2^{\Omega(S)}$. Reingold, Rothblum and Rothblum [16] gave a doubly efficient protocol for any time T with constantly many rounds of communication, but is only efficient for the verifier when T is polynomial.

► **Theorem 8 (RRR Protocol)**. *For any constant $\delta > 0$, and integers S and T computable in time $T^{O(\delta)}S^2$, and $T = \Omega(n)$ we have*

$$\text{TISP}[T, S] \subseteq \text{ITIME}^1[O(n\text{polylog}(T) + T^{O(\delta)}S^2), T^{1+O(\delta)}\text{poly}(S)].$$

Further the prover only sends $(\frac{1}{8})^{O(1/\delta)}$ messages to the verifier.

The specific, S^2 power in the verifier time comes from a note by Goldreich [7], confirmed by the authors of [16]. Our result gives a more efficient verifier ($\log(T)S$ vs $T^\delta S^2$), but a less efficient prover ($2^{O(S)}$ vs $\text{poly}(T)$). We note the result in the [16] paper allows sub constant δ , but is complex and can not give a verifier with time $\text{poly}(\log(T)S)$.

There has been work on other notions of verifier efficiency in interactive protocols. Goldwasser, Gutfreund, Healy, Kaufman and Rothblum [8] studied the computation depth required by verifiers. They showed that for any k round interactive proof, there is a $k + O(1)$ round interactive proof where the computation of the verifier during each round is in **NC**⁰. But the total time to evaluate the new verifier (or the total verifier circuit size) is greater than the verifier time of the original protocol.

² Achieving this claimed performance with GKR is not trivial, but we believe it can be done.

■ **Table 1** Comparison of different protocol times, with polylogarithmic factors omitted. The first three columns are verifier times for three special cases and the last column is prover time, which is the same for all three cases. RRR does not work for nondeterministic algorithms.

	TISP	BPTISP	NTISP	Prover
Shamir	$\log(T)(S + n)$	$\log(T)^2 S + \log(T)n$	$\log(T)^2(S + n)$	$2^{O(\log(T)(S+n))}$
GKR	$\log(T)S^2 + n$	$\log(T)S^2 + n$	$\log(T)S^2 + n$	$2^{O(S)}$
RRR	$T^{O(\delta)}S^2 + n$	$T^{O(\delta)}S^2 + n$	-	$T^{1+O(\delta)}S^{O(1)}$
Ours	$\log(T)S + n$	$\log(T)S + n$	$\log(T)^2 S + n$	$2^{O(S)}$

2 Proof Idea

Our protocol uses sum check [11], but unlike Shamir, Shen, and Meir [21, 22, 12], we do not reduce to a quantified Boolean formula first. Instead, we reduce to matrix exponentiation. This gives us **IP = PSPACE** from sum check with fewer steps.

Our protocol improves over Shamir's in two important ways. One, it gives better results when $S = o(n)$. This is due to a better arithmetization of RAM algorithms directly, allowing us to leave the input out of the algorithm's state. This more efficient arithmetization is the primary reason we use the RAM model.

The other is a more efficient prover. This comes from a different reduction to matrix exponentiation instead of quantified formulas. Our reductions gives a more efficient prover algorithm. We will now describe our protocol.

For an algorithm A running on input x in time T and space S , we want to know whether $A(x) = 1$, or if $A(x) = 0$. In an interactive proof, there is a computationally bounded verifier, V , who wants to know $A(x)$, and an unbounded, untrusted prover, P , who wants to convince V of a value for $A(x)$, but may lie. Verifier V can ask many questions to prover P and P answers instantly. If V was deterministic, this would just be NP , so V has access to randomness that P cannot predict.

Our proof is based on low degree polynomials over some field \mathbb{F} of size $\mathbf{poly}(S)$. For space, we only prove the case for deterministic algorithms. See [6] for the full version of this paper. We separate our main proof into several ideas:

1. Computation Graph and Matrix Exponentiation.

Let M be the adjacency matrix of the computation graph of algorithm A on input x . That is, $M_{a,b} = 1$ if and only if when A on input x is in state a , it transitions to state b in one step. See that $(M^T)_{a,b} = 1$ (where M^T is M to the T th power, not the transpose of M) if and only if $A(x)$ is in state b after T steps when starting in state a .

2. Arithmetization.

For any field \mathbb{F} , a verifier can efficiently compute the multilinear extension of M , which we denote $\widehat{M} : \mathbb{F}^S \times \mathbb{F}^S \rightarrow \mathbb{F}$. That is, \widehat{M} is multilinear and for $a, b \in \{0, 1\}^S$, we have $\widehat{M}(a, b) = M_{a,b}$. See Definition 17.

The goal is to give a protocol for reducing a claim that $\alpha = \widehat{M}^2(a, b)$, to a claim that $\alpha' = \widehat{M}(a', b')$. If we can construct such a protocol, applying it $\log(T)$ times reduces our claim that M^T ends in an accept state to a claim about \widehat{M} , which the verifier can compute itself.

3. Sum check.

We can write \widehat{M}^2 , the multilinear extension of M^2 , in terms of \widehat{M} as $\widehat{M}^2(a, b) = \sum_{c \in \{0, 1\}^S} \widehat{M}(a, c)\widehat{M}(c, b)$. Using the sum check protocol from [11], we can reduce a claim that $\alpha = \widehat{M}^2(a, b)$ for some $\alpha \in \mathbb{F}$ and $a, b \in \mathbb{F}^S$ to the claim that $\beta = \widehat{M}(a, c)\widehat{M}(c, b)$ for some $\beta \in \mathbb{F}$ and a random $c \in \mathbb{F}^S$.

4. Product reduction.

There is a trick used in [9] that reduces the claim that $\beta = \widehat{M}(a, c)\widehat{M}(c, b)$ to the claim that $\alpha' = \widehat{M}(a', b')$ for some $\alpha' \in \mathbb{F}$ and $a', b' \in \mathbb{F}^S$.

5. Repeated square rooting.

Applying the above protocols $\log(T)$ times, we reduce a claim that M^T has a one in the transition from the start state to the end state, to a claim that $\alpha' = \widehat{M}(a', b')$ which the verifier can calculate and check itself.

Now we explain each of these ideas in a little more detail.

2.1 Computation Graphs

For an algorithm A running in space S on input x , its computation graph G has as vertices S bit states and as edges the state transitions for A on input x . That is, there is an edge from state a to b if and only if when A on input x is in state a , after one step, A is in state b . Let M be the adjacency matrix of G . If A runs in T steps starting in state a , and b is the accept state, then A accepts if and only if $(M^T)_{a,b}$ is one.

2.2 Arithmetization and Low Degree Extensions

For this strategy to work, the verifier needs to be able to compute some error correcting code of the computation, to compare against the claim of the prover. This will be a multilinear extension of M .

A key difference between our arithmetization and Shamir's is that our model of computation is the RAM model, and Shamir's is a single tape Turing machine. So inherent to Shamir's protocol, the state must include the input, but ours does not. This is why we get better results for $S = o(n)$. Further, by arithmetizing the transition function directly instead of reducing to a formula first, we are able to get the stronger multilinear extension instead of just a low degree extension.

We say that $\widehat{\phi} : \mathbb{F}^S \times \mathbb{F}^S \rightarrow \mathbb{F}$ is the multilinear extension of $\phi : \{0, 1\} \times \{0, 1\} \rightarrow \mathbb{F}$ if $\widehat{\phi}$ has degree at most 1 in each variable and for all $a, b \in \{0, 1\}^S$, $\phi(a, b) = \widehat{\phi}(a, b)$. Note multilinear extensions are unique (see Lemma 16). If M is a $2^S \times 2^S$ matrix with $M_{a,b} = \phi(a, b)$, then we define the multilinear extension of M , denoted $\widehat{M} : \mathbb{F}^S \times \mathbb{F}^S \rightarrow \mathbb{F}$, by $\widehat{M}(a, b) = \widehat{\phi}(a, b)$. See Definition 17.

In general, it may be hard to compute low degree extensions. For general functions, it requires reading the whole size 2^{2S} truth table! But many classes of functions have low degree extensions that are easy to compute. Specifically, it is easy to compute the low degree extension of the state transition function of RAM algorithms.

► **Lemma 9 (Algorithm Arithmetization).** *Let A be a nondeterministic RAM algorithm running in space S and time T on length n inputs, and x be an input with $|x| = n$. Define M to be the $2^S \times 2^S$ matrix such that for any two states $a, b \in \{0, 1\}^S$, we have $M_{a,b} = 1$ if when A is running on input x is in state a , then b as a valid transition, and $M_{a,b} = 0$ otherwise.*

Then we can compute the multilinear extension of M (\widehat{M} in Definition 17) in time $\tilde{O}(\log(|\mathbb{F}|)(n + S))$.

The idea is to sum over all instructions the multilinear polynomial identifying that instruction being the current instruction, times the multilinear polynomial of that instruction being performed. Since algorithm A is constant, there are only constantly many instructions we could be on. It is easy to compute multilinear extensions of register operations, and easy to compute multilinear extensions of loading or storing from main memory. See Section 4.3 for details on how to perform the arithmetization.

2.3 Sum Check

The key part of our protocol is the sum check from LFKN [11]. Suppose we have a degree d polynomial over S variables: $q : \mathbb{F}^S \rightarrow \mathbb{F}$. Then sum check allows an efficient verifier with access to randomness to verify the sum of q on all Boolean inputs with the help of an untrusted prover.

► **Lemma 10** (Sum Check Protocol). *Let S be an integer, d be an integer, p be a prime, and \mathbb{F} be a field with characteristic p where $|\mathbb{F}| \geq d + 1$. Suppose $q : \mathbb{F}^S \rightarrow \mathbb{F}$ be a polynomial with degree at most d in each variable individually. For a multi-linear polynomial, $d = 1$.*

Then there is a interactive protocol with verifier V and prover P such that on input $\alpha \in \mathbb{F}$ behaves in the following way:

Completeness: *If $\alpha = \sum_{a \in \{0,1\}^S} q(a)$, then when V interacts with P , verifier V outputs some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Soundness: *If $\alpha \neq \sum_{a \in \{0,1\}^S} q(a)$, then for any prover P' , when V interacts with P' , with probability at most $\frac{dS}{|\mathbb{F}|}$ will V output some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Verifier Time: V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(Sd)$.

Prover Time: P runs in time $\tilde{O}(\log(|\mathbb{F}|))\mathbf{poly}(d)2^S$ using $O(d2^S)$ oracle queries to q .

The idea of sum check is to choose the elements of a' one at a time, and ask about univariate polynomials that are partial sums: filled in with the chosen parts of a' , leaving one variable unfixed, and summed over the rest of the variables. These are error correcting codes, and one is calculated from the next, which can be checked. Due to Schwartz–Zippel, with high probability, if the claimed equality is wrong, each of these low degree polynomials from the proof must be wrong to not be caught by the verifier. See Appendix A for a proof.

Now our protocol wants to reduce a claim that $\alpha = \widehat{M}^2(a, b)$ to a claim that $\alpha' = \widehat{M}^2(a', b')$. When we inspect $\widehat{M}^2(a, b)$, we see that $\widehat{M}^2(a, b) = \sum_{c \in \{0,1\}^S} \widehat{M}(a, c)\widehat{M}(c, a)$. Then considering the degree 2 in each variable polynomial $q(c) = \widehat{M}(a, c)\widehat{M}(c, a)$, sum check reduces the claim that $\alpha = \widehat{M}^2(a, b)$ to the claim that $\beta = \widehat{M}(a, c)\widehat{M}(c, a)$ for some c . This is very nearly what we want.

2.4 Product Reduction

Now that we have a claim that $\beta = \widehat{M}(a, c)\widehat{M}(c, a)$, our verifier needs to check this. One obvious idea is to ask the prover for both $\alpha_1 = \widehat{M}(a, c)$, and $\alpha_2 = \widehat{M}(c, a)$. Then the prover could prove both of these statements separately. But this would double the number of claims the verifier must check everytime we reduce \widehat{M}^2 to \widehat{M} . Since we have to do this $\log(T)$ times, the verifier would need to check T claims! So this cannot be done.

Instead, the verifier needs to reduce to a claim about \widehat{M} at a single point to avoid this. The idea is to take a line, $\psi : \mathbb{F} \rightarrow (\mathbb{F}^S \times \mathbb{F}^S)$, that passes through both (a, c) and (c, a) , and ask the prover for $\widehat{M} \circ \psi$. The function $\widehat{M} \circ \psi$ will have degree $2S$, since \widehat{M} is multilinear. [9] uses a similar trick.

Now $\widehat{M} \circ \psi(0) = \widehat{M}(a, c)$ and $\widehat{M} \circ \psi(1) = \widehat{M}(c, a)$. If the $\beta \neq \widehat{M}(a, c)\widehat{M}(c, a)$, then the prover cannot be honest about $\widehat{M} \circ \psi$, or the verifier will reject. Now the verifier chooses a $d \in \mathbb{F}$, and wants to know $\alpha' = \widehat{M}(\psi(d))$. If the prover was honest, then this will give us that at $(a', b') = \psi(d)$, we have $\alpha' = \widehat{M}(a', b')$. But if the prover lied, then by Schwartz–Zippel, with high probability, $\alpha' \neq \widehat{M}(a', b')$. See Lemma 18 for more details.

2.5 Protocol For Deterministic Algorithms

Using sum check and product reduction, there is a protocol that takes a claim that $\alpha = \widehat{M^2}(a, b)$ to the claim that $\alpha' = \widehat{M}(a', b')$. Note that this protocol works with ANY matrix M , not just the M from the adjacency matrix of the computation graph.

In particular, our prover can start by claiming that for start state a , and end state b , that $\widehat{M^T}(a, b) = 1$. The verifier itself can confirm that a is the start state, and b is an accept, or reject state. If A is deterministic, then A accepts x if and only $\widehat{M^T}(a, b) = 1$.

Using the matrix squared to matrix protocol, there is a protocol to reduce the claim that $\widehat{M^T}(a, b) = 1$ to the claim that $\widehat{M^{T/2}}(a_1, b_1) = \alpha_1$. Then to the claim that $\widehat{M^{T/4}}(a_2, b_2) = \alpha_2$. After repeating this $t = \log(T)$ times, we reduce to the claim that $\widehat{M}(a_t, b_t) = \alpha_t$. But this can be directly checked by the verifier.

3 Preliminaries

We use RAM algorithms with registers and a program as our model of computation. This is used for our efficient arithmetization. But the verifier is very simple and can be implemented efficiently in other common models of computation, like multi-tape Turing machines. We may assume that on accepting or rejecting, our machines instantly clear their states to some canonical accept or reject state. We also assume that $S = \Omega(\log(n))$, otherwise our algorithm can't read its entire input.

We think of our algorithms as defining invalid and valid state transitions. For deterministic algorithms, every state has exactly one valid transition. Then a deterministic algorithm accepts if and only if there is some sequence of memory states such that every transition is valid, the first is the start state, and the final is the accept state.

3.1 Complexity Classes

We use \tilde{O} to suppress poly logarithmic factors.

► **Definition 11** (\tilde{O}). For $f, g : \mathbb{N} \rightarrow \mathbb{N}$, we say $f(n) = \tilde{O}(g(n))$ if and only if for some constant k , $f(n) = O(g(n) \log(g(n))^k)$.

We focus on languages with simultaneous time and space constraints.

► **Definition 12 (TISP)**. For functions $T, S : \mathbb{N} \rightarrow \mathbb{N}$, we say language L is in **TISP** $[T, S]$ if there is an algorithm, A , running in time T and space S that recognizes L .

For space, we focus on deterministic algorithms, see the full paper [6] for the randomized and nondeterministic cases.

In an interactive protocol for some function f , we want our verifier to output $f(x)$ with high probability if the prover is honest, and outputs the wrong answer with low probability regardless of the prover. Our verifier can either reject or output some value. We use double sided completeness, since that is what we prove.

Let us formally define the interaction of a protocol.

► **Definition 13 (Interaction Between Verifier and Prover (Int))**. Let Σ be a alphabet and \perp be a symbol not in Σ . Let V be a RAM machine with access to randomness, that can make oracle queries, and V outputs one of $\Sigma \cup \{\perp\}$. Let P' be any function, and x be an input.

Now we define the interaction of V and P' on input x . For all i , define y_i to be V 's i th oracle query given its first $i - 1$ queries were answered with z_1, \dots, z_{i-1} and define $z_i = P'(x, y_1, \dots, y_i)$.

Define the output of V when interacting with P' , $\text{Int}(V, P', x)$, as the output of V on input x when its oracle queries are answered by z_1, z_2, \dots .

Now we define interactive time.

► **Definition 14** (Interactive Time (**ITIME**)). *Let Σ be an alphabet. If for function $f : \{0, 1\}^* \rightarrow \Sigma$, soundness $s \in [0, 1]$, completeness $c \in [0, 1]$, verifier V and prover P we have*

Completeness: $\Pr[\text{Int}(V, P, x) = f(x)] \geq c$, and

Soundness: for any function P' we have $\Pr[\text{Int}(V, P', x) \notin \{f(x), \perp\}] \leq s$,

then we say V and P are an interactive protocol for f with soundness s and completeness c .

If in addition L is a language with $f(x) = 1_{x \in L}$, verifier V runs in time T_V , soundness $s < \frac{1}{3}$, and completeness $c > \frac{2}{3}$, then $L \in \mathbf{ITIME}[T_V]$. If P is also computable by an algorithm running in time T_P , we say $L \in \mathbf{ITIME}[T_V, T_P]$. Finally, if completeness $c = 1$, then we say the protocol has perfect completeness and $L \in \mathbf{ITIME}^1[T_V, T_P]$.

3.2 Multilinear Extensions

Sum check is generally used on Boolean functions. So first the Boolean function must be converted to a low degree polynomial, a technique broadly called arithmetization. Arithmetization is an important part of sum check [11]. Shamir [21] arithmetized Boolean formulas. Boolean formulas of size C have a low degree extension of total degree C . The idea is to rewrite every $\alpha \wedge \beta$ into $\alpha \cdot \beta$, every $\alpha \vee \beta$ as $\alpha + \beta - \alpha \cdot \beta$ and every $\neg \alpha$ as $1 - \alpha$.

This indeed gives a low degree polynomial for formulas, but we use a very strong type of low degree polynomial: multilinear extensions.

► **Definition 15** (Multilinear Extension). *For a field \mathbb{F} , integer S and a function $\phi : \{0, 1\}^S \rightarrow \mathbb{F}$, we define the multilinear extension of ϕ as the polynomial $\widehat{\phi} : \mathbb{F}^S \rightarrow \mathbb{F}$ that is degree 1 in any individual variable such that for all $a \in \{0, 1\}^S$ we have $\phi(a) = \widehat{\phi}(a)$.*

One useful property of multilinear extensions is that unlike low degree extensions, multilinear extensions are unique. See the full version for a proof [6].

► **Lemma 16** (Multilinear Extension Exist and are Unique). *For a field \mathbb{F} , integer S and a function $\phi : \{0, 1\}^S \rightarrow \mathbb{F}$, there exists $\widehat{\phi}$ that is a multilinear extension of ϕ . Further, $\widehat{\phi}$ is unique.*

For notation, we will also define the multilinear extension of matrices as the multilinear extension of the function which indexes into that matrix.

► **Definition 17** (Matrix Multilinear Extension). *Let S be an integer, \mathbb{F} a field, and M be a $2^S \times 2^S$ matrix containing elements in \mathbb{F} . Identify an element $x \in \{0, 1\}^S$ with an element of $[2^S]$ by interpreting x as a binary number.*

Then define the multilinear extension of M to be the function $\widehat{M} : \mathbb{F}^S \rightarrow \mathbb{F}^S \rightarrow \mathbb{F}$ such that \widehat{M} is multilinear and for $a, b \in \{0, 1\}^S$ we have $\widehat{M}(a, b) = M_{a,b}$. See that \widehat{M} exists and is unique by Lemma 16.

4 Efficient IP for TISP

The main building block is a protocol to reduce a statement about a matrix squared to a statement about the matrix itself. Then I show how to combine this with arithmetization to give a protocol for bounded space.

The first step in the matrix squared to matrix reduction is the sum check protocol Lemma 10. See Appendix A for details about the sum check protocol. This reduces a statement about \widehat{M}^2 to a statement about a product of \widehat{M} .

4.1 Product Reduction

After the sum check, we need to reduce a statement about a product of \widehat{M} evaluated at two points to a statement about \widehat{M} evaluated at one point.

► **Lemma 18** (Product Reduction). *Let S be an integer, d be an integer, p be a prime, and \mathbb{F} be a field with characteristic p where $|\mathbb{F}| \geq d + 1$. Suppose $q : \mathbb{F}^S \rightarrow \mathbb{F}$ is a polynomial with total degree at most d . For a multi-linear polynomial, $d = S$.*

Then there is a interactive protocol with verifier V and prover P such that on input $\alpha \in \mathbb{F}$, $a, b \in \mathbb{F}^S$ behaves in the following way:

Completeness: *If $\alpha = q(a)q(b)$, then when V interacts with P , verifier V outputs some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Soundness: *If $\alpha \neq q(a)q(b)$, then for any prover P' , when V interacts with P' , with probability at most $\frac{d}{|\mathbb{F}|}$ will V output some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(S + d)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))\text{Spoly}(d)$ using $d + 1$ oracle calls to q .*

Proof. The idea is to choose a line, $\psi : \mathbb{F} \rightarrow \mathbb{F}^S$, such that $\psi(0) = a$ and $\psi(1) = b$. That is, $\psi(c) = (1 - c)a + cb$. Then the verifier asks the prover for the degree d polynomial $g : \mathbb{F} \rightarrow \mathbb{F}$ defined by $g(c) = q(\psi(c))$. For such a g , see that $q(a)q(b) = g(0)g(1)$.

Let g' be the degree d polynomial returned by the prover. If $\alpha \neq g'(0)g'(1)$, the verifier rejects. Otherwise, the verifier chooses a random $c \in \mathbb{F}$, sets $a' = \psi(c)$ and sets $\alpha' = g'(c)$.

Now I show this protocol has the desired properties.

Completeness: If $\alpha = q(a)q(b)$, an honest prover responds with $g' = g$, so

$$\alpha = q(a)q(b) = q(\psi(0))q(\psi(1)) = g(0)g(1) = g'(0)g'(1).$$

Then the verifier chooses c and $\alpha' = g'(c) = g(c) = q(\psi(c)) = q(a')$.

Soundness: If $\alpha \neq q(a)q(b)$, then if $g' = g$, we have $g'(0)g'(1) = q(a)q(b) \neq \alpha$, so the verifier rejects. Otherwise, $g \neq g'$, so with probability at most $\frac{d}{|\mathbb{F}|}$ does $g(c) = g'(c)$. If $g(c) \neq g'(c)$, then $\alpha' = g'(c) \neq g(c) = q(\psi(c)) = q(a')$. Thus with probability at most $\frac{d}{|\mathbb{F}|}$ does $q(a') = \alpha'$.

Verifier Time: V computes g' three times, and each time this only takes $O(d)$ field operations. V computes $\psi(c)$ once, which takes $O(S)$ field operations. Every field operation takes $\tilde{O}(\log(|\mathbb{F}|))$ time.

Prover Time: P queries q at $d + 1$ points to calculate q . These $d + 1$ query locations can be calculated in $O(dS)$ field operations by evaluating ψ . Then the coefficients for g can be calculated in time polynomial time in d using gaussian elimination. ◀

4.2 Matrix Squared To Matrix Reduction

Now for the main lemma used in our proof. First, we need a lemma that shows \widehat{M}^2 can be written as a simple function of \widehat{M} .

► **Lemma 19** (\widehat{M}^2 is a sum of products of \widehat{M}). *Let S be an integer and \mathbb{F} be a field. Suppose M is a $2^S \times 2^S$ matrix containing values in \mathbb{F}_p . Then for any $a, b \in \mathbb{F}^S$,*

$$\widehat{M}^2(a, b) = \sum_{c \in \{0,1\}^S} \widehat{M}(a, c)\widehat{M}(c, b),$$

where \widehat{M} and \widehat{M}^2 is as defined in Definition 17.

Proof. For notation, define ψ by $\psi(a, b) = \sum_{c \in \{0,1\}^S} \widehat{M}(a, c) \widehat{M}(c, b)$ so that we are trying to prove that $\widehat{M}^2 = \psi$. By Lemma 16, $\widehat{M}^2 = \psi$ if and only if ψ is multilinear and they agree on all binary inputs.

To see that ψ is multilinear, we show that for any $c \in \{0,1\}^S$ we have $\widehat{M}(a, c) \widehat{M}(c, b)$ is multilinear as a function of a and b . But they are since \widehat{M} is multilinear and the two calls to \widehat{M} don't share any variables. Thus ψ is the sum of multilinear polynomials, and is thus multilinear itself.

To see that ψ agrees with \widehat{M}^2 on binary elements, take any $a, b \in \{0,1\}^S$. Then we have

$$\widehat{M}^2(a, b) = (M^2)_{a,b} = \sum_{c \in \{0,1\}^S} M_{a,c} M_{c,b} = \sum_{c \in \{0,1\}^S} \widehat{M}(a, c) \widehat{M}(c, b) = \psi(a, b).$$

So ψ must be the unique multilinear polynomial agreeing with \widehat{M}^2 on all binary inputs, which is \widehat{M}^2 itself. \blacktriangleleft

► **Lemma 20** (Matrix Squared to Matrix Protocol). ³ Let S be an integer, p be a prime, and \mathbb{F} be a field with characteristic p and $|\mathbb{F}| > 2S + 1$. Suppose M is a $2^S \times 2^S$ matrix containing values in \mathbb{F}_p . Define \widehat{M} as in Definition 17.

Then there is a interactive protocol with verifier V and prover P such that on input $a, b \in \mathbb{F}^S$ and $\alpha \in \mathbb{F}$ behaves in the following way:

Completeness: If $\alpha = \widehat{M}^2(a, b)$, then when V interacts with P , verifier V outputs some $a', b' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = \widehat{M}(a', b')$.

Soundness: If $\alpha \neq \widehat{M}^2(a, b)$, then for any prover P' , when V interacts with P' , with probability at most $\frac{4S}{|\mathbb{F}|}$ will V output $a', b' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = \widehat{M}(a', b')$.

Verifier Time: V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Prover Time: P runs in time $\tilde{O}(\log(|\mathbb{F}|))2^S$ when given oracle access to \widehat{M} .

Proof. The protocol is a sum check Lemma 10 followed by a product reduction Lemma 18. From Lemma 19, see that $\alpha = \widehat{M}^2(a, b) = \sum_{c \in \{0,1\}^S} \widehat{M}(a, c) \widehat{M}(c, b)$.

The first sum check either fails, or gives the claim that for some $c \in \mathbb{F}^S$, and $\beta \in \mathbb{F}$ that $\beta = \widehat{M}(a, c) \widehat{M}(c, b)$. Then the product reduction fails, or gives the claim that for some $a', b' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ that $\alpha' = \widehat{M}(a', b')$.

Now we can check that this protocol gives the desired results.

Completeness: Suppose $\alpha = \widehat{M}^2(a, b)$. Then by completeness of the sum check, $\beta = \widehat{M}(a, c) \widehat{M}(c, b)$. Then by completeness of the product reduction, $\alpha' = \widehat{M}(a', b')$.

Soundness: Suppose $\alpha \neq \widehat{M}^2(a, b)$.

Let $q : \mathbb{F}^S \rightarrow \mathbb{F}^S$ be defined by $q(c) = \widehat{M}(a, c) \widehat{M}(c, a)$. That is, q is the function sum check is performed on. For any variable y , we know q is degree at most 2 in y as the product of two degree one polynomials in y .

Then by soundness of sum check, with probability at most $\frac{2S}{|\mathbb{F}|}$ will the verifier output β and c such that $\beta = \widehat{M}(a, c) \widehat{M}(c, b)$.

See that since \widehat{M} is multilinear, \widehat{M} has total degree at most $2S$. If $\beta \neq \widehat{M}(a, c) \widehat{M}(c, b)$, then by the soundness of the product reduction, with probability at most $\frac{2S}{|\mathbb{F}|}$ does $\alpha' = \widehat{M}(a', b')$. Thus by a union bound, with probability at most $\frac{4S}{|\mathbb{F}|}$ does $\alpha' = \widehat{M}(a', b')$.

Verifier Time: V takes the time of the sum check, plus the time of product reduction, which is $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Prover Time: P runs in the max of the time for the sum check and the time for the product reduction, which is $\tilde{O}(\log(|\mathbb{F}|))2^S$ when given oracle access to \widehat{M} . \blacktriangleleft

³ Lemma 20 was first proven by Thaler [24].

4.3 Arithmetization

To use this matrix square reduction, we need to actually compute the multilinear extension of the adjacency matrix of the computation graph. The multilinear extensions of many simple functions are efficient to calculate, for instance, the equality function. See Appendix B for examples.

Now we show how to calculate the multilinear extension of a RAM computer's state transition function.

► **Lemma 9** (Algorithm Arithmetization). *Let A be a nondeterministic RAM algorithm running in space S and time T on length n inputs, and x be an input with $|x| = n$. Define M to be the $2^S \times 2^S$ matrix such that for any two states $a, b \in \{0, 1\}^S$, we have $M_{a,b} = 1$ if when A is running on input x is in state a , then b as a valid transition, and $M_{a,b} = 0$ otherwise.*

Then we can compute the multilinear extension of M (\widehat{M} in Definition 17) in time $\tilde{O}(\log(|\mathbb{F}|)(n + S))$.

Proof. The state of any RAM algorithm has 3 parts:

- The instruction pointer into some fixed program.
- Constant number of registers of $O(\log(n + S))$ length for performing register register operations. We assume these operations are simple operations like move, bit shift, bitwise or, bitwise and, addition, and conditional jumps in the program counter.
- S bits of memory that can be changed by the load and store commands.

Then we decompose any state, $a \in \{0, 1\}^S$ into three parts $a = (p, r, m)$ where p are constantly many bits for the instruction pointer, r is $O(\log(n + S))$ many bits for the registers, and m is $O(S)$ many bits for main memory.

Suppose there are I instructions. For $i \in [I]$, let P_i be the multilinear function that identifies if the program is at instruction i . Let Q_i be the multilinear function of whether the current state on instruction i yields the next state. Then a formula for \widehat{M} is given by

$$\widehat{M}((p_0, r_0, m_0), (p_1, r_1, m_1)) = \sum_{i \in [I]} P_i(p_0) Q_i(r_0, m_0, p_1, r_1, m_1).$$

Thus if we can calculate each Q_i , we can calculate \widehat{M} . The exact possible instructions depend on our choice of instruction set, but we cover the main ones here.

- Register to Register Arithmetic.
Register register operations (like bit shifts, additions, etc) can be computed very efficiently. So Q_i just is the product of: the appropriate registers being updated correctly, the instruction pointer being incremented by one, all other registers being equal in r_0 and r_1 , and m_0 and m_1 being equal.
- Conditionals.
For conditional jumps, first compute equality of the state before and after, besides the program counter and the condition bit. Then multiply that by the sum over the multilinear extension of the condition bit leading to the correct location of the program counter.
- Load Input into a Register.
A load from input instruction operates on two registers, one to store the input, and a pointer to the location in the input. The rest of the state is just an equality. Suppose the location you want to retrieve is indicated by $a_1, \dots, a_{\log(n)}$, and you want to store it in variable b_1 . Then the extension of this part is

$$\sum_{i \in \{0, 1\}^{\log(n)}} \prod_{j \in [\log(n)]} (a_j i_j + (1 - a_j)(1 - i_j))(x_i b_1 + (1 - x_i)(1 - b_1)).$$

This can be computed efficiently, see the full version of the paper [6].

- Load Memory into a Register.

Basically uses the same formula as above, but uses a $\log(S)$ bit address and uses memory instead of the input. That is, replace x with m_0 . One slight technicality is that we must also assert that $m_0 = m_1$. That is, main memory doesn't change. Specifically, we calculate

$$\sum_{i \in \{0,1\}^{\log(S)}} \left(\prod_{j \in [\log(S)]} (a_j i_j + (1 - a_j)(1 - i_j)) \right) \\ ((m_0)_i (m_1)_i b_1 + (1 - (m_0)_i)(1 - (m_1)_i)(1 - b_1)) \\ \prod_{k \in \{0,1\}^{\log(S)} \setminus \{i\}} ((m_0)_k (m_1)_k + (1 - (m_0)_k)(1 - (m_1)_k)).$$

- Store Register into Memory.

This is essentially the same as loading, except instead of saying the future register should be equal to the current memory, we instead say the future memory is equal to the current register.

Thus each Q_i can be efficiently calculated in $O(S + n)$ field operations, so \widehat{M} can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$. ◀

4.4 Protocol for TISP

Now we give our protocol for deterministic algorithms.

► **Theorem 1** (Efficient Interactive Protocol For TISP). *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$ with $S = \Omega(\log(n))$. Then*

$$\text{TISP}[T, S] \subseteq \text{ITIME}^1[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

Proof. We start by outlining how to convert acceptance to a matrix problem, then we show how to apply Lemma 20 to solve that. Let a be the canonical starting state of algorithm A , and b the canonical accept state.

Take T to be a power of two so that $T = 2^t$. If T is not a power of 2, we can take T to be the smallest power of two greater than the original T . Let k be the smallest integer so that $2^k > 12 \log(T)S$. Let $q = 2^k$ and \mathbb{F} be the field with q elements. Note that $|\mathbb{F}| \leq 24 \log(T)S$.

Let M be the adjacency matrix for the computation graph of A on input x so that for any two states, a' and b' , we have $M_{a',b'} = 1$ if A on input x starting in state a' can be b' after one step, and $M_{a',b'} = 0$ otherwise. For any matrix M' , let \widehat{M}' be the multilinear extension of M' so that for all binary inputs a' and b' we have $\widehat{M}'(a', b') = M'_{a',b'}$.

Observe that $M_{a',b'}^{2^i} = 1$ if when A on input x is in state a' after 2^i steps is in state b' , and $M_{a',b'}^{2^i} = 0$ otherwise. Thus our verifier wants to output $M^T(a, b)$, or $\widehat{M}^{2^t}(a, b)$.

In the full protocol, the prover first provides the verifier with a candidate $\alpha \in \{0, 1\}$ with the claim that $\alpha = \widehat{M}^{2^t}(a, b)$. Now we use Lemma 20 t times to get the claim that for some $\alpha' \in \mathbb{F}$ and some $a', b' \in \mathbb{F}^S$ we have $\alpha' = \widehat{M}(a', b')$. Finally, the verifier uses Lemma 9 to calculate $\widehat{M}(a', b')$ and compare it with α' .

Completeness: For an honest prover, indeed $\alpha = \widehat{M}^{2^t}(a, b)$. Let $\alpha_0 = \alpha$, $a_0 = a$ and $b_0 = b$. By induction and completeness of Lemma 20, for every $i \in [0, t - 1]$ we have $\alpha_i = M^{2^{t-i}}(a_i, b_i)$, and our protocol gives an α_{i+1} , a_{i+1} , and b_{i+1} such that $\alpha_{i+1} = M^{2^{t-(i+1)}}(a_{i+1}, b_{i+1})$. Thus $\alpha_t = \widehat{M}(a_t, b_t)$. Thus the verifier check whether $\alpha_t = \widehat{M}(a_t, b_t)$ succeeds, and the verifier outputs α .

14:14 More Verifier Efficient Interactive Protocols for Bounded Space

Soundness: If $\alpha = \widehat{M}^{2^t}(a, b)$, the verifier either outputs α or rejects, either satisfies our assumption. So suppose $\alpha \neq \widehat{M}^{2^t}(a, b)$. Let $\alpha_0 = \alpha$, $a_0 = a$ and $b_0 = b$. By induction and soundness of Lemma 20, for every $i \in [0, t-1]$ if the verifier hasn't rejected and $\alpha_i \neq \widehat{M}^{2^{t-i}}(a_i, b_i)$, the probability the verifier does not reject and $\alpha_{i+1} = \widehat{M}^{2^{t-(i+1)}}(a_{i+1}, b_{i+1})$ is at most $\frac{4S}{|\mathbb{F}|}$. By a union bound, the probability that the verifier does not reject and for any i we have $\alpha_i = \widehat{M}^{2^{t-i}}(a_i, b_i)$ is at most $\frac{4S \log(T)}{|\mathbb{F}|} \leq \frac{1}{3}$.

In particular, the probability the verifier does not reject and $\alpha_t = \widehat{M}(a_t, b_t)$ is at most $\frac{1}{3}$. See that if $\alpha_t \neq \widehat{M}(a_t, b_t)$, then the verifier rejects. So the probability the verifier does not reject is at most $\frac{1}{3}$.

Verifier Time: This verifier takes time $\tilde{O}(\log(|\mathbb{F}|))S \log(T)$ to run Lemma 20 t times plus $\tilde{O}(\log(|\mathbb{F}|))O(S+n)$ to calculate $\widehat{M}(a_t, b_t)$ (using Lemma 9), so in total takes time $\tilde{O}(\log(|\mathbb{F}|))(S \log(T) + n) = \tilde{O}(\log(S))O(\log(T))S + n$.

Prover Time: First, the prover calculates M^{2^i} in time $\tilde{O}(\log(|\mathbb{F}|))S2^{3S}$ for every $i \in [t]$.

Then we can query the multilinear extension of these matrices, which is \widehat{M}^{2^i} , at any location in time $\tilde{O}(\log(|\mathbb{F}|))2^{2S}$ with the formula given in Lemma 16.

Finally, given oracle access to each \widehat{M}^{2^i} , the prover in Lemma 20 only takes time $\tilde{O}(\log(|\mathbb{F}|))2^S$. Thus of any call to Lemma 20 only takes time **poly** $(\log(|\mathbb{F}|))2^{3S}$. Thus the prover runs in time $2^{O(S)}$. ◀

We can extend this result into multi bit outputs by storing the output in the final end state and asking the prover for the end state first. Or to be more efficient when outputs are larger than S , we can first ask for the output. Then include the output in the input and change the algorithm to instead verify the output is correct. Then run the protocol on this new verification algorithm instead. This allows us to verify program outputs larger than S more efficiently.

For the randomized and nondeterministic cases, see the full paper [6].

5 Open Problems

Here are some open problems.

1. Could one remove the dependence on $\log(T)$? Is it true that, for $S = \omega(n)$, we have $\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S)]$?

This would prove an equivalence, up to polylogarithmic factors, between $\mathbf{SPACE}[S]$ and $\mathbf{ITIME}[S]$. Our recent work, [13], showed a similar equivalence between $\mathbf{NTIME}[T]$ and languages verified by \mathbf{PCPs} with $\log(T)$ time verifiers, for $\log(T) = \Omega(n)$.

2. Is there a strong hierarchy theorem for interactive time? Can we show that for any T we have $\mathbf{ITIME}[\tilde{O}(T)] \not\subseteq \mathbf{ITIME}[T]$? Here $\mathbf{ITIME}[T]$ is the class of languages with an interactive protocol whose verifiers run in time T .

Using Theorem 1 gives $\mathbf{ITIME}^1[T] \subseteq \mathbf{SPACE}[O(T)] \subseteq \mathbf{ITIME}^1[\tilde{O}(T^2)]$. This gives a hierarchy theorem, $\mathbf{ITIME}^1[\tilde{O}(T^2)] \not\subseteq \mathbf{ITIME}^1[O(T)]$, by using the space hierarchy theorem. Improving the interactive protocols for bounded space is one way to give a stronger hierarchy.

3. Can interactive protocols for \mathbf{NTISP} be as efficient as those for \mathbf{TISP} ?

For $S = \Omega(n)$, we get $\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^2)]$, but only show $\mathbf{NSPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^3)]$. Does nondeterministic space require more verifier time than deterministic space?

4. Can we get double efficiency with a similar verifier time? Is it true that $\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\mathbf{polylog}(T)S, \mathbf{poly}(T)]$?

This would make the verification of polynomial time algorithms only take as long (up to polylogarithmic factors) as the space of those algorithms, with a proof that still only takes polynomial time. This would make directly using interactive proofs for delegating computation more practical.

References

- 1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.
- 2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306.
- 3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, January 1998. doi:10.1145/273865.273901.
- 4 L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 16–25 vol.1, 1990. doi:10.1109/FSCS.1990.89520.
- 5 Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2022. doi:10.1109/FOCS52979.2021.00021.
- 6 Joshua Cook. More verifier efficient interactive protocols for bounded space, 2022. URL: <https://eccc.weizmann.ac.il/report/2022/093/>.
- 7 Oded Goldreich. On doubly-efficient interactive proof systems, 2018. URL: <https://www.wisdom.weizmann.ac.il/~oded/de-ip.html>.
- 8 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 440–449. Association for Computing Machinery, 2007. doi:10.1145/1250790.1250855.
- 9 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), September 2015. doi:10.1145/2699436.
- 10 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi:10.1137/0217058.
- 11 C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 2–10 vol.1, 1990. doi:10.1109/FSCS.1990.89518.
- 12 Or Meir. IP = PSPACE using error-correcting codes. *SIAM Journal on Computing*, 42(1):380–403, 2013. doi:10.1137/110829660.
- 13 Dana Moshkovitz and Joshua Cook. Tighter $ma/1$ circuit lower bounds from verifier efficient pcps for pspace, 2022. URL: <https://eccc.weizmann.ac.il/report/2022/014/>.
- 14 Cody Murray and Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: An easy witness lemma for np and nqp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 890–901. Association for Computing Machinery, 2018. doi:10.1145/3188745.3188910.
- 15 Noam Nisan. Pseudorandom generators for space-bounded computations. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 204–212. Association for Computing Machinery, 1990. doi:10.1145/100216.100242.
- 16 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 49–62. Association for Computing Machinery, 2016. doi:10.1145/2897518.2897652.

- 17 Noga Ron-Zewi and Ron D. Rothblum. Proving as fast as computing: Succinct arguments with constant prover overhead. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1353–1363. Association for Computing Machinery, 2022. doi:10.1145/3519935.3519956.
- 18 Michael Saks and Shiyu Zhou. $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. *J. Comput. Syst. Sci.*, 58(2):376–403, April 1999. doi:10.1006/jcss.1998.1616.
- 19 Rahul Santhanam. Circuit lower bounds for merlin-arthur classes. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 275–283. Association for Computing Machinery, 2007. doi:10.1145/1250790.1250832.
- 20 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, April 1970. doi:10.1016/S0022-0000(70)80006-X.
- 21 Adi Shamir. $IP = PSPACE$. *J. ACM*, 39(4):869–877, October 1992. doi:10.1145/146585.146609.
- 22 A. Shen. $IP = SPACE$: Simplified Proof. *J. ACM*, 39(4):878–880, 1992. doi:10.1145/146585.146613.
- 23 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988. doi:10.1007/BF00299636.
- 24 Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 71–89, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

A Sum Check

Sum check [11] takes a claim that for some $\alpha \in \mathbb{F}$, we have $\alpha = \sum_{s \in \{0,1\}^S} q(s)$, and reduces it to the claim that for some $\beta \in \mathbb{F}$ and for some random $r \in \mathbb{F}^S$ we have $\beta = q(r)$.

► **Lemma 10 (Sum Check Protocol).** *Let S be an integer, d be an integer, p be a prime, and \mathbb{F} be a field with characteristic p where $|\mathbb{F}| \geq d + 1$. Suppose $q : \mathbb{F}^S \rightarrow \mathbb{F}$ be a polynomial with degree at most d in each variable individually. For a multi-linear polynomial, $d = 1$.*

Then there is a interactive protocol with verifier V and prover P such that on input $\alpha \in \mathbb{F}$ behaves in the following way:

Completeness: *If $\alpha = \sum_{a \in \{0,1\}^S} q(a)$, then when V interacts with P , verifier V outputs some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Soundness: *If $\alpha \neq \sum_{a \in \{0,1\}^S} q(a)$, then for any prover P' , when V interacts with P' , with probability at most $\frac{dS}{|\mathbb{F}|}$ will V output some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(Sd)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))\mathbf{poly}(d)2^S$ using $O(d2^S)$ oracle queries to q .*

Sketch. The idea is to choose the random field elements for a' one at a time and consider the partial sums. Let $\alpha_0 = \alpha$. For every i , we want to reduce a claim that

$$\alpha_i = \sum_{a \in \{0,1\}^{S-i}} p(a'_1, \dots, a'_i, a),$$

to a claim that

$$\alpha_{i+1} = \sum_{a \in \{0,1\}^{S-(i+1)}} p(a'_1, \dots, a'_i, a'_{i+1}, a).$$

The observation is that if we define $g_i : \mathbb{F} \rightarrow \mathbb{F}$ by

$$g_i(x) = \sum_{a \in \{0,1\}^{S-(i+1)}} p(a'_1, \dots, a'_i, x, a)$$

then g_i is a low degree polynomial. The verifier asks for g_i , and checks if $g_i(0) + g_i(1) = \alpha_i$. If it doesn't, the verifier knows the prover lied and rejects. Otherwise, it chooses r_{i+1} and sets $\alpha_{i+1} = g_i(r_{i+1})$.

If g_i is correct, then $\alpha_{i+1} = \sum_{s' \in \{0,1\}^{S-(i+1)}} p(r_1, \dots, r_i, r_{i+1}, s')$. Thus for an honest prover, this equality will hold for every i , and the completeness holds.

Now if at any step $\alpha_i \neq \sum_{s \in \{0,1\}^{S-i}} p(r_1, \dots, r_i, s)$, the prover cannot provide the correct g_i , or the verifier will see that $g_i(0) + g_i(1) \neq \alpha_i$ and reject. But if g_i is incorrect, by Schwartz-Zippel, the probability the provided g_i agrees with the true g_i is $\frac{d}{|\mathbb{F}|}$. If they disagree at r_{i+1} , then $\alpha_{i+1} \neq \sum_{s' \in \{0,1\}^{S-(i+1)}} p(r_1, \dots, r_i, r_{i+1}, s')$. So by a union bound, the probability the verifier ever has a correct α_i is at most $\frac{Sd}{|\mathbb{F}|}$. ◀

This is a commonly taught protocol, and can be found in “Computational Complexity: A Modern Approach” [1].

B Arithmetization Examples

Here are some examples of efficient multilinear extensions.

► **Lemma 21** (Equality has Efficient Multilinear Extension). *Let $\phi : \{0, 1\}^S \times \{0, 1\}^S \rightarrow \{0, 1\}$ be the Boolean function such that $\phi(a, b) = 1$ if and only if $a = b$.*

Then for any field \mathbb{F} , the multilinear extension of ϕ , denoted $\widehat{\phi}$, can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Proof. One can write the formula for $\widehat{\phi}$ as

$$\widehat{\phi}(a, b) = \prod_{i \in [S]} (a_i b_i + (1 - a_i)(1 - b_i)).$$

To see the above equality, see that when a and b are restricted to binary variables, the right hand side is one if and only if for every i we have $a_i = b_i$. Further, the right hand side is degree one any individual a_i or b_i .

Further, it can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$ since each field operation only takes time $\tilde{O}(\log(|\mathbb{F}|))$, and as written, the function only needs $O(S)$ field operations. ◀

► **Remark 22.** One can easily extend this to check equality of 3, 4, or any arbitrary number of variables. Extra equality checks are often easy to add. To assert $d = e$ in most formulas, just replace every occurrence of d with de and $(1 - d)$ with $(1 - d)(1 - e)$. This works as long as the formula can be rewritten as a sum of products where every product has either d or $1 - d$ exactly once.

This is true for equality, cyclic bit shifting, and even addition.

Similarly, inequality, or any constant bit shift can also be computed efficiently. For a more complex example, we can efficiently compute the multilinear extension of binary addition.

► **Lemma 23** (Binary Addition has Efficient Multilinear Extension). *Let $\phi_S : \{0, 1\}^S \times \{0, 1\}^S \times \{0, 1\}^S \rightarrow \{0, 1\}$ be the Boolean function such that $\phi(a, b, c) = 1$ if and only if $a + b = c$ where addition is binary and we ignore the final carry.*

Then for any field \mathbb{F} , the multilinear extension of ϕ_S , $\widehat{\phi}_S$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Proof. We prove this recursively. Define $\psi_S : \{0, 1\}^S \times \{0, 1\}^S \times \{0, 1\}^S \rightarrow \{0, 1\}$ as the Boolean function such that $\psi(a, b, c) = 1$ if and only if $a + b + 1 = c$ where addition is binary and we ignore the final carry. Let $\widehat{\psi}_S$ be the multilinear extension of ψ_S . The idea is to implement a ripple carry adder.

14:18 More Verifier Efficient Interactive Protocols for Bounded Space

Now we will compute $\widehat{\psi}_S$ and $\widehat{\phi}_S$ by induction. See that

$$\begin{aligned}\widehat{\phi}_1(a, b, c) &= (a(1-b) + b(1-a))c + (ab + (1-a)(1-b))(1-c) \\ \widehat{\psi}_1(a, b, c) &= (a(1-b) + b(1-a))(1-c) + (ab + (1-a)(1-b))c.\end{aligned}$$

These are multilinear as written, and correctly compute ψ_1 and ϕ_1 when given binary inputs. Each of these only require a constant number of field operations.

Assume for $a', b', c' \in \mathbb{F}^{S-1}$, we have computed $\widehat{\phi}_{S-1}(a', b', c')$ and $\widehat{\psi}_{S-1}(a', b', c')$. We show how to calculate $\widehat{\psi}_S$ and $\widehat{\phi}_S$. For $a, b, c \in \mathbb{F}$, I claim that

$$\begin{aligned}\widehat{\phi}_S((a, a'), (b, b'), (c, c')) &= \widehat{\phi}_{S-1}(a', b', c')(1-a)(1-b)(1-c) \\ &\quad + \widehat{\phi}_{S-1}(a', b', c')(a(1-b) + (1-a)b)c \\ &\quad + \widehat{\psi}_{S-1}(a', b', c')ab(1-c) \\ \widehat{\psi}_S((a, a'), (b, b'), (c, c')) &= \widehat{\phi}_{S-1}(a', b', c')(1-a)(1-b)c \\ &\quad + \widehat{\psi}_{S-1}(a', b', c')(a(1-b) + (1-a)b)(1-c) \\ &\quad + \widehat{\psi}_{S-1}(a', b', c')abc.\end{aligned}$$

These are multilinear by induction. For binary inputs, this implements a ripple carry adder. For instance, if $a = b = 0$, then $\widehat{\phi}((a, a'), (b, b'), (c, c'))$ is one if and only if $c = 0$ and, by induction, $a' + b' = c'$. If a and b are zero, then c should be 0 and there is no carry. The rest of the cases can be checked similarly

Given that $\widehat{\psi}_{S-1}$ and $\widehat{\phi}_{S-1}$ were already calculated, it only requires constantly many more field operations to calculate $\widehat{\psi}_S$ and $\widehat{\phi}_S$. Thus, $\widehat{\psi}_S$ and $\widehat{\phi}_S$ only require $O(S)$ field operations. Thus $\widehat{\psi}_S$ only takes time $\tilde{O}(\log(|\mathbb{F}|))S$ to calculate. ◀

Note, we could modify this to give a multilinear extension of the function $\phi' : \{0, 1\}^{5S} \rightarrow \{0, 1\}$ that not only checks if $a + b = c$, but also whether $a = d$ and $b = e$, as described in Remark 22.

One can compute the multilinear extensions of other register, register operations in a simple register RAM model computer.

Improved Quantum Query Upper Bounds Based on Classical Decision Trees

Arjan Cornelissen  

Institute for Logic, Language, and Computation, University of Amsterdam, The Netherlands
QuSoft, Amsterdam, The Netherlands

Nikhil S. Mande  

QuSoft, Amsterdam, The Netherlands
CWI, Amsterdam, The Netherlands

Subhasree Patro  

QuSoft, Amsterdam, The Netherlands
CWI Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Abstract

We consider the following question in query complexity: Given a classical query algorithm in the form of a decision tree, when does there exist a quantum query algorithm with a speed-up (i.e., that makes fewer queries) over the classical one? We provide a general construction based on the structure of the underlying decision tree, and prove that this can give us an up-to-quadratic quantum speed-up in the number of queries. In particular, our results give a bounded-error quantum query algorithm of cost $O(\sqrt{s})$ to compute a Boolean function (more generally, a relation) that can be computed by a classical (even randomized) decision tree of size s . This recovers an $O(\sqrt{n})$ algorithm for the Search problem, for example.

Lin and Lin [Theory of Computing'16] and Beigi and Taghavi [Quantum'20] showed results of a similar flavor. Their upper bounds are in terms of a quantity which we call the “guessing complexity” of a decision tree. We identify that the guessing complexity of a decision tree equals its *rank*, a notion introduced by Ehrenfeucht and Haussler [Information and Computation'89] in the context of learning theory. This answers a question posed by Lin and Lin, who asked whether the guessing complexity of a decision tree is related to any measure studied in classical complexity theory. We also show a polynomial separation between rank and its natural randomized analog for the complete binary AND-OR tree.

Beigi and Taghavi constructed span programs and dual adversary solutions for Boolean functions given classical decision trees computing them and an assignment of non-negative weights to edges of the tree. We explore the effect of changing these weights on the resulting span program complexity and objective value of the dual adversary bound, and capture the best possible weighting scheme by an optimization program. We exhibit a solution to this program and argue its optimality from first principles. We also exhibit decision trees for which our bounds are strictly stronger than those of Lin and Lin, and Beigi and Taghavi. This answers a question of Beigi and Taghavi, who asked whether different weighting schemes in their construction could yield better upper bounds.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Query Complexity, Decision Trees, Decision Tree Rank

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.15

Related Version *Full Version:* <https://arxiv.org/abs/2203.02968> [12]

Funding *Nikhil S. Mande:* Supported by the Dutch Research Council (NWO/OCW), as part of the Quantum Software Consortium programme (project number 024.003.037), and through QuantERA ERA-NET Cofund project QuantAlgo (project number 680-91-034, ended in December 2021).

Subhasree Patro: Mainly supported by the Robert Bosch Stiftung, also additionally supported by NWO Gravitation grants NETWORKS and QSC, and through QuantERA ERA-NET Cofund project QuantAlgo (project number 680-91-034, ended in December 2021).



© Arjan Cornelissen, Nikhil S. Mande, and Subhasree Patro;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 15; pp. 15:1–15:22



Leibniz International Proceedings in Informatics

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

Acknowledgements We thank Ronald de Wolf for useful comments.

1 Introduction

In this paper, we address the following question: given a classical algorithm performing a task, along with a description of the algorithm, when can one turn it into a quantum algorithm and obtain a speed-up in the process? Of specific interest to us in this paper is the case when the classical algorithm can be efficiently represented by a decision tree. For instance, consider the problem of identifying the first marked item in a list, say x of n items, each of which may or may not be marked. The input is initially unknown, and one has access to it via an *oracle*. On being queried $i \in [n]$, the oracle returns whether or not x_i is marked. Moreover, queries can be made in superposition. The goal is to minimize the number of queries in the worst case and output the correct answer with high probability for every possible input list. This problem is closely related to the Search problem, and is known to admit a quadratic quantum speed-up (its classical query complexity is $\Theta(n)$ and quantum query complexity is $\Theta(\sqrt{n})$) [16, 21, 23]. It is easy to construct a classical decision tree (in fact, a decision list) of depth n and size (which is the number of nodes) $2n + 1$ that solves the above-mentioned problem. In view of the quadratic speed-up that quantum query algorithms can achieve for this problem, this raises the following natural question: Given a classical decision tree of size s that computes a function, is there a bounded-error quantum query algorithm of cost $O(\sqrt{s})$ that solves the same function? Among other results, we answer this in the affirmative (see Corollary 9). We obtain our quantum query upper bounds by constructing explicit *span programs* and *dual adversary solutions* by exploiting structure of the initial classical decision tree. In the discussions below, let $Q_\varepsilon(f)$ be the ε -error quantum query complexity of f . When $\varepsilon = 1/3$, we drop the subscript and call $Q(f)$ the bounded-error quantum query complexity of f .

1.1 Span Programs

Span programs are a computational model introduced by Karchmer and Wigderson [20]. Roughly speaking, a span program defines a function depending on whether or not the “target vector” of an input is in the span of its associated “input vectors”. Span programs were first used in the context of quantum query complexity by Reichardt and Špalek [30], and it is known that span programs characterize bounded-error quantum query complexity of Boolean functions up to a constant factor [29, 22]. Span programs have been used to design quantum algorithms for various graph problems such as *st*-connectivity [9], cycle detection and bipartiteness testing [3, 11], graph connectivity [18], and has been also used for problems such as formula evaluation [30, 28, 19]. Recently, Beigi and Taghavi [4] defined a variant of span programs (non-binary span programs with orthogonal inputs, abbreviated NBSPwOI) for showing upper bounds on the quantum query complexity of non-Boolean input/output functions $f : [\ell]^n \rightarrow [m]$. Moreover in a follow-up work [5], they also use non-binary span programs for showing upper bounds for a variety of graph problems, for example, the maximum bipartite matching problem.

1.2 Dual Adversary Bound

The general adversary bound for Boolean functions f was developed by Høyer, Lee and Špalek [17], and they showed that this quantity gives a lower bound on the bounded-error quantum query complexity of f . It was eventually shown that this bound actually

characterizes the bounded-error quantum query complexity of f up to a constant factor [29, 22]. The general adversary bound can be expressed as a semidefinite program that admits a dual formulation. Thus, feasible solutions to the dual adversary program yield quantum query upper bounds. Quantum query algorithms have been developed by explicitly constructing dual adversary solutions, for example for the well-studied k -distinctness problem [7], S -isomorphism and hidden subgroup problems [8], gapped group testing [2], etc.

1.3 Related Works

Lin and Lin [23] showed how a classical algorithm computing a function $f : D_f \rightarrow \mathcal{R}$ with $D_f \subseteq \{0, 1\}^n$ and \mathcal{R} as an arbitrarily large finite set, equipped with an efficient “guessing scheme”, could be used to construct a faster quantum query algorithm computing f . Moreover, their results apply to the setting where $f \subseteq \{0, 1\} \times \mathcal{R}$ is a relation. A deterministic algorithm computing a relation f takes an input $x \in \{0, 1\}^n$ and outputs a value b such that $(x, b) \in f$. More precisely, they showed the following:¹

► **Theorem 1** (Lin and Lin [23, Theorem 5.4]). *Let $f \subseteq \{0, 1\}^n \times \mathcal{R}$ be a relation. Let \mathcal{A} be a deterministic algorithm computing f that makes at most T queries. Let $x_{p_1}, \dots, x_{p_{\tilde{T}(x)}}$ be the query results of \mathcal{A} on an a priori unknown input string x . For $x \in \{0, 1\}^n$, let $\tilde{T}(x) \leq T$ denote the number of queries that \mathcal{A} makes on input x . Suppose there is another deterministic algorithm \mathcal{G} which takes as input $b_1, \dots, b_{t-1} \in \{0, 1\}^{t-1}$ for any $t \in [T]$, and outputs a guess for the next query result of \mathcal{A} . Assume that \mathcal{G} makes at most G mistakes for all x . That is,*

$$\forall x \in \{0, 1\}^n, \quad \sum_{t=1}^{\tilde{T}(x)} |\mathcal{G}(x_1, \dots, x_{p_{t-1}}) - x_{p_t}| \leq G.$$

Then $Q(f) = O(\sqrt{TG})$.

Lin and Lin provided two proofs of the above theorem, one of which involved constructing an explicit quantum query algorithm. This algorithm is iterative, and works as follows: In the i 'th iteration, use a modified version of Grover's search algorithm to find the next mistake that \mathcal{G} makes. Since the algorithm uses at most T queries and makes at most G mistakes on any input, the quantum query complexity of the final algorithm can be shown to be $O(\sqrt{TG})$ using the Cauchy-Schwarz inequality.²

Recently, Beigi and Taghavi [5] gave an alternate proof of Theorem 1 using the framework of *non-binary span programs* introduced by them in [4]. Using this they showed that a similar statement also holds for functions $f : [\ell]^n \rightarrow \mathcal{R}$ with non-binary inputs and outputs. A sketch of their proof is as follows: Given an assignment of real weights to the edges of the given decision tree computing f , they construct a dual adversary solution and span program. The complexity of the resultant query algorithm is a function of the weights assigned to the edges. They propose a particular weighting scheme based on an edge-coloring (guessing algorithm) of the given decision tree. For Boolean functions, the quantum query complexity upper bound they obtain matches the bound of Lin and Lin (Theorem 1), which is $O(\sqrt{TG})$, where T denotes the depth and G the *guessing complexity* (see Definition 11) of the underlying decision tree, respectively.

¹ For ease of readability, we state the theorem for deterministic decision trees. Lin and Lin actually showed a stronger statement that holds even if one considers randomized decision trees and randomized guessing algorithms.

² We skip some crucial details here, such as the cost of the modified Grover's search algorithm, and an essential step that uses span programs to avoid a logarithmic overhead that error reduction would have incurred. Techniques that address both of the above-mentioned steps are due to Kothari [21].

In a follow-up work [6], conditional on some constraints, Beigi, Taghavi and Tajdini implement the span-program-based algorithm of [5] in a time-efficient manner. In another work [32], Taghavi uses non-binary span programs to give a tight quantum query algorithm for the oracle identification problem, simplifying an earlier algorithm due to Kothari [21].

1.4 Our Contributions

The *rank* of a decision tree is a combinatorial measure introduced by Ehrenfeucht and Haussler [14] in the context of learning theory, formally defined as follows.

► **Definition 2** (Decision tree rank and randomized rank). *Let \mathcal{T} be a binary decision tree. Define the rank of \mathcal{T} recursively as follows: For a leaf node a , define $\text{rank}(a) = 0$. For an internal node u with children v, w , define*

$$\text{rank}(u) = \begin{cases} \max\{\text{rank}(v), \text{rank}(w)\} & \text{if } \text{rank}(v) \neq \text{rank}(w) \\ \text{rank}(v) + 1 & \text{if } \text{rank}(v) = \text{rank}(w). \end{cases}$$

Define $\text{rank}(\mathcal{T})$ to be the rank of the root of \mathcal{T} . Define the randomized rank of a randomized decision tree to be the maximum rank of a deterministic decision tree in its support.

► **Definition 3** (Rank of a Boolean function). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function. Define the rank of f , which we denote by $\text{rank}(f)$, by*

$$\text{rank}(f) = \min_{\mathcal{T}: \mathcal{T} \text{ computes } f} \text{rank}(\mathcal{T}).$$

Analogously define the randomized rank of f , which we denote by $\text{rrank}(f)$, to be the minimum randomized rank of a randomized decision tree that computes f to error $1/3$.

We observe here that the guessing complexity of a deterministic decision tree equals its rank (see Claim 12). This answers a question of Lin and Lin [23, page 4], where the authors asked if G is related to any combinatorial measure studied in classical decision-tree complexity. The rank of a function (which is the minimum rank of a decision tree computing it) can be exponentially smaller than the function's certificate complexity, sensitivity, block sensitivity, and even (exact or approximate) polynomial degree, as can be easily witnessed by the OR function. See, for example, [13] for more relationships between rank and other combinatorial measures of Boolean functions. In view of the above-mentioned equivalence of G and decision tree rank, Theorem 1 has a clean equivalent formulation as follows.

► **Theorem 4.** *Let \mathcal{T} be a decision tree computing a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ with depth T . Then $Q(f) = O(\sqrt{\text{rank}(\mathcal{T})T})$.*

Randomized rank, which we denote by rrank , is a natural probabilistic analog of rank. This exactly captures the notion of the randomized analog of the value of G in Theorem 1. It is easy to show with this definition that our proof of Theorem 4 also holds with respect to randomized decision trees and randomized rank: sample a decision tree from the support of \mathcal{T} according to its underlying distribution, and run a $(1/10)$ -error³ quantum query algorithm of cost $O(\sqrt{\text{rank}(\mathcal{T})T})$ from Theorem 4 on the resultant tree. The success probability is at least $(9/10) \cdot (2/3) = 3/5$. Thus we obtain the following easy-to-state reformulation of [23, Theorem 5.4].

³ The success probability of the algorithm in Theorem 4 can be boosted by repeating the algorithm a large constant number of times and then outputting the majority of the outcomes.

► **Theorem 5.** *Let \mathcal{T} be a randomized decision tree computing a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ with depth T . Then $Q_{2/5}(f) = O(\sqrt{\text{rrank}(\mathcal{T})T})$.*

Thus, up to a small loss in the success probability, upper bounds obtained from Theorem 5 are strictly stronger than those obtained from Theorem 4 for relations whose randomized rank is much smaller than their rank. Hence a natural question is if this can be the case.

It is easy to exhibit maximal separations between rank and randomized rank for partial functions. One such separation is witnessed by the Approximate-Majority function, which is the function that outputs 0 if the Hamming weight of an n -bit input is less than $n/3$ and outputs 1 if the Hamming weight of the input is more than $2n/3$. It is easy to show an $\Omega(n)$ lower bound on its rank and an $O(1)$ upper bound on its randomized rank. Whether or not such a separation holds for total Boolean functions is not so clear. We show a polynomial separation for the complete binary AND-OR tree. This is the first of our main theorems.

► **Theorem 6.** *For the complete binary AND-OR tree $f : \{0, 1\}^n \rightarrow \{0, 1\}$,*

$$\text{rrank}(f) = O\left(\text{rank}(f)^{\log \frac{1+\sqrt{33}}{4}}\right) \approx O\left(\text{rank}(f)^{.753\dots}\right).$$

To prove this theorem, we show a rank lower bound of $\Omega(n)$ using known connections between rank and Prover-Delayer games [27], and the randomized-rank upper bound immediately follows from an upper bound of $O\left(n^{\log \frac{1+\sqrt{33}}{4}}\right)$ on its randomized decision-tree complexity [31].

We conjecture that rank and randomized rank are polynomially related for all total Boolean functions. Note that techniques used to prove the analogous polynomial equivalence for deterministic and randomized query complexities [26] (also see [10]) cannot work since they use intermediate measures such as certificate complexity, sensitivity and block sensitivity, all of which are maximal for the OR function even though its rank is 1.

Beigi and Taghavi [5, Section 6] asked if one could improve their results by using different choices of weights in their constructions of span programs and dual adversary vectors. We answer this question in the affirmative by providing a weighting scheme that improves upon their bounds. By a careful analysis, we argue that the optimizing the dual adversary bound and the witness complexity of Beigi and Taghavi’s span program with variable weights is a minimization program with constraints linear in the variables and inverses of the variables.

► **Definition 7** (Weight optimization program). *For a decision tree \mathcal{T} , define its weight optimization program by the minimization problem with constraints outlined in Program 1 (see Appendix A.1 for relevant definitions). Let $\text{OPT}_{\mathcal{T}}$ denote the optimum of this program.*

■ **Program 1** The *weight optimization program* capturing the weight assignment to edges of \mathcal{T} that optimizes the witness complexity of the NBSPwOI and dual adversary vector constructions of Beigi and Taghavi (see Section 3).

Variables	$\{W_e : e \text{ is an edge in } \mathcal{T}\}, \alpha, \beta$		
Minimize	$\sqrt{\alpha\beta}$		
s.t.	$\sum_{e \in \bar{P}} W_e$	$\leq \alpha,$	for all paths $P \in P(\mathcal{T})$
	$\sum_{e \in P} \frac{1}{W_e}$	$\leq \beta,$	for all paths $P \in P(\mathcal{T})$
	W_e	$> 0,$	for all edges e in \mathcal{T}
	α, β	$> 0.$	

For a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ and a deterministic decision tree \mathcal{T} computing it, let \tilde{f} be the function that takes an n -bit string x as input, and outputs the leaf of \mathcal{T} reached on input x . The following is our second main theorem.

► **Theorem 8.** *Let $f \subseteq \{0, 1\}^n \times \mathcal{R}$ be a relation and let \mathcal{T} be a decision tree computing f . Then $Q(\tilde{f}) = O(\text{OPT}_{\mathcal{T}})$.*

We give a recursively-defined weighting scheme that achieves equality for all the constraints in Program 1 and argue from first principles that our solution is optimal (see Theorem 26), thus subsuming Theorem 1. Combined with the earlier results from Beigi and Taghavi and using our recursive expression for $\text{OPT}_{\mathcal{T}}$, this gives us the following corollary, giving a new way to bound $\text{OPT}_{\mathcal{T}}$ and thus $Q(\tilde{f})$ from above.

► **Corollary 9.** *Let $f \subseteq \{0, 1\}^n \times \mathcal{R}$ be a relation and let \mathcal{T} be a deterministic decision tree computing it, weighted with the canonical weight assignment as defined in Definition 23. Then, the quantum query complexity of \tilde{f} (in fact $\text{OPT}_{\mathcal{T}}$) satisfies the following two bounds:*

1. *The rank-depth bound: $Q(\tilde{f}) = O\left(\sqrt{\text{rank}(\mathcal{T})\text{depth}(\mathcal{T})}\right)$.*
2. *The size bound: $Q(\tilde{f}) = O\left(\sqrt{\text{DTSize}(\mathcal{T})}\right)$ where $\text{DTSize}(\mathcal{T})$ denotes the number of nodes in \mathcal{T} .*

Using standard arguments to deal with the case when \mathcal{T} is a randomized decision tree, this gives an upper bound on the bounded-error quantum query complexity of a function in terms of its randomized decision-tree size complexity, as well as in terms of its randomized rank and depth (see Corollary 27).

It was shown by Reichardt [29] that the quantum query complexity of evaluating Boolean formulas of size s is $O(\sqrt{s})$. In particular, this implies the size bound in Corollary 9 for Boolean functions f since the formula size of a Boolean function is bounded above by a constant times its decision-tree size (see the full version of this paper [12, Claim A.2]). Not only does our bound also hold for *relations*, but the query algorithm we obtain is actually an algorithm for \tilde{f} when the underlying tree is deterministic. While this yields trivial bounds for most relations (since almost all Boolean functions have super-polynomial decision-tree size complexity, for example, while $Q(f)$ is at most n), it recovers the $O(\sqrt{n})$ bound for the Search problem [16], for example. We also exhibit a family of decision trees for which the size bound is strictly stronger than the rank-depth bound of Lin and Lin, and Beigi and Taghavi (see Figure 3).

1.5 Organization

In Section 2, we discuss the rank of decision trees, and prove that it is equal to guessing complexity. Furthermore, we prove a separation between rank and randomized rank for the complete binary AND-OR tree. In Appendix B we discuss how to construct a span program and a dual adversary solution for a relation f by assigning weights to edges of a classical decision tree computing f , and we capture the weighting scheme in an optimization program. In Section 3 we prove Theorem 8. Finally, in Section 4, we exhibit a solution to Program 1 and argue its optimality from first principles. We refer the reader to the appendices for preliminaries and missing proofs.

2 Decision Tree Rank

In this section, we first rephrase Theorem 1 in terms of a measure of decision trees which we term “guessing complexity”. This reformulation was essentially done by Beigi and Taghavi [5, Section 3]. We then show that the guessing complexity of a decision tree equals its rank, proving Theorem 4. Finally, we show a polynomial separation between rank and randomized rank for the complete binary AND-OR tree.

2.1 Guessing Complexity and Rank

► **Definition 10** (G-coloring [5, Definition 1]). A *G-coloring* of a decision tree \mathcal{T} is a coloring of its edges by two colors black and red, in such a way that any vertex of \mathcal{T} has at most one outgoing edge with black color.

► **Definition 11** (Guessing Complexity). Let \mathcal{T} be a decision tree and let $P(\mathcal{T})$ denote the set of root to leaf paths in \mathcal{T} . Define the guessing complexity of \mathcal{T} , which we denote by $G(\mathcal{T})$, by $G(\mathcal{T}) = \min_{G\text{-colorings of } \mathcal{T}} \max_{P \in P(\mathcal{T})} \text{number of red edges on } P$.

▷ **Claim 12.** Let \mathcal{T} be a decision tree. Then $G(\mathcal{T}) = \text{rank}(\mathcal{T})$.

Proof. Let v , v_L and v_R be the root of \mathcal{T} , and the left and right children of v , respectively. Let \mathcal{T}_L and \mathcal{T}_R denote the subtrees of \mathcal{T} rooted at v_L and v_R , respectively.

Consider a *G-coloring* of \mathcal{T} . This naturally induces a *G-coloring* of \mathcal{T}_L and \mathcal{T}_R . We consider two cases:

- $G(\mathcal{T}_L) = G(\mathcal{T}_R) = k$, say. One of the edges (v, v_L) or (v, v_R) must be colored red. Assume without loss of generality that (v, v_L) is the red edge. Since we assumed $G(\mathcal{T}_L) = k$, \mathcal{T}_L contains a path with at least k red edges under the *G-coloring* induced from the given *G-coloring* of \mathcal{T} . But this induces a path in \mathcal{T} with $k + 1$ red edges, and hence $G(\mathcal{T}) \geq G(\mathcal{T}_L) + 1$.
- If $G(\mathcal{T}_L) \neq G(\mathcal{T}_R)$, we have $G(\mathcal{T}) \geq \max\{G(\mathcal{T}_L), G(\mathcal{T}_R)\}$, witnessed by the *G-colorings* induced on \mathcal{T}_L and \mathcal{T}_R by the *G-coloring* of \mathcal{T} .

In the other direction, we construct an optimal *G-coloring* of \mathcal{T} given optimal *G-colorings* of \mathcal{T}_L and \mathcal{T}_R . The edges of \mathcal{T}_L and \mathcal{T}_R in \mathcal{T} are colored exactly as they are in the given optimal *G-colorings* of them. It remains to assign colors to the two remaining edges (v, v_L) and (v, v_R) . We again have two cases:

- $G(\mathcal{T}_L) = G(\mathcal{T}_R) = k$, say. Arbitrarily color one of the edges (v, v_L) and (v, v_R) (say, (v, v_L)) red, and color the other edge black. The maximum number of red edges on a path has increased by 1. Thus, $G(\mathcal{T}) \leq G(\mathcal{T}_L) + 1$.
- $G(\mathcal{T}_L) > G(\mathcal{T}_R)$, say (the other case follows a similar argument). Color the edge (v, v_L) black and (v, v_R) red. Thus, the maximum number of red edges on a path in \mathcal{T} equals $\max\{G(\mathcal{T}_L), G(\mathcal{T}_R) + 1\} = G(\mathcal{T}_L)$.

Thus, we have

$$G(\mathcal{T}) = \begin{cases} \max\{G(\mathcal{T}_L), G(\mathcal{T}_R)\} & \text{if } G(\mathcal{T}_L) \neq G(\mathcal{T}_R) \\ G(\mathcal{T}_L) + 1 & \text{if } G(\mathcal{T}_L) = G(\mathcal{T}_R), \end{cases}$$

The measure $\text{rank}(\mathcal{T})$ is defined exactly as the above (Definition 2), proving the claim. ◁

The guessing algorithm \mathcal{G} in Theorem 1 corresponds to a natural *G-coloring* of \mathcal{T} of cost G : for each internal vertex, color the guessed edge black and the other edge red. Thus, Theorem 4 immediately follows from Claim 12 and Theorem 1.

Proof of Theorem 5. An algorithm for f is as follows: sample a decision tree from the support of \mathcal{T} according to its underlying distribution, and run a 9/10-error quantum query algorithm from Theorem 4 on the resultant tree.⁴ The cost of this algorithm is $O(\sqrt{\text{rank}(\mathcal{T})T})$ and the success probability is at least $(9/10) \cdot (2/3) = 3/5$ for all inputs $x \in D_f$. ◀

⁴ This is possible since the deterministic decision trees in the support of \mathcal{T} compute *functions*, which admit efficient error reduction with a constant overhead in query complexity by standard techniques (run the algorithm from Theorem 4 a large constant many times and return the majority output).

15:8 Improved Quantum Query Upper Bounds Based on Classical Decision Trees

The rank of a decision tree essentially captures the largest depth of a binary subtree of the original tree. Thus, the rank of a tree is bounded from above by the logarithm of the size of the tree.

► **Observation 13** ([14, Lemma 1]). *Let \mathcal{T} be a deterministic decision tree of size s . Then $\text{rank}(\mathcal{T}) \leq \log(s + 1) - 1$.*

Along with this observation and the simple observation that the depth of a decision tree is at most its size, Theorem 5 yields the following statement.

► **Theorem 14.** *Let \mathcal{T} be a randomized decision tree of size s that computes a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$. Then $\mathbf{Q}_{2/5}(f) = O(\sqrt{s \log s})$.*

Note here that it suffices to prove that $\mathbf{Q}(\tilde{f}) = O(\sqrt{s \log s})$ where s is the size of a *deterministic* decision tree computing f , since standard techniques and error reduction yield the required bound for randomized trees. In the full version of this paper [12, Appendix B], we show an explicit NBSPW_{OI} and dual adversary solution witnessing the same bound. In Sections 3 and 4 we show an explicit NBSPW_{OI} and dual adversary solution witnessing a stronger bound without the logarithmic factor. We choose to still give the weaker bound in [12, Appendix B] as the weighting scheme seems considerably different from that in Section 4, and the weights are also efficiently computable.

We note here the equivalence of the rank of a Boolean function and the value of an associated Prover-Delayer game introduced by Pudlák and Impagliazzo [27]. We use this equivalence in the next part of this section to show that the rank of the complete binary AND-OR tree is polynomially larger than its randomized rank.

The game is played between two players: the Prover and the Delayer, who construct a partial assignment, say $\rho \in \{0, 1, \perp\}^n$, in rounds. To begin with the assignment is empty, i.e., $\rho = \perp^n$. In a round, the Prover queries an index $i \in [n]$ for which the value x_i is not set in ρ (i.e., $\rho_i = \perp$). The Delayer either answers $x_i = 0$ or $x_i = 1$, or defers the choice to the Prover. In the latter case, the Delayer scores a point. The game ends when the Prover knows the value of the function, i.e., when $f|_\rho$ is a constant function. The value of the game, $\text{val}(f)$, is the maximum number of points the Delayer can score regardless of the Prover's strategy. The following result is implicit in [27] (also see [13, Theorem 3.1] for an explicit statement and proof).

▷ **Claim 15.** Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a (possibly partial) Boolean function. Then $\text{rank}(f) = \text{val}(f)$.

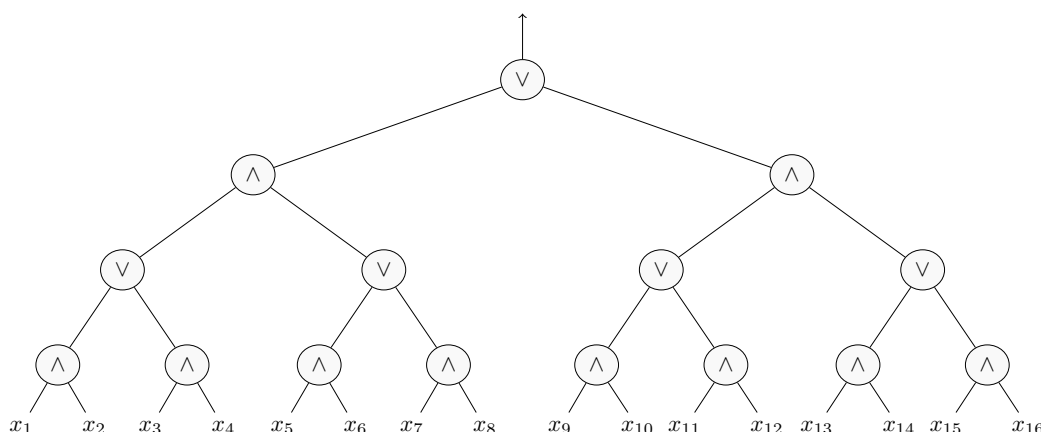
2.2 A Separation Between Rank and Randomized Rank

We first note that there can be maximal separations between rank and randomized rank if we consider partial functions, i.e., functions defined only on a subset of all possible inputs. This is witnessed by the well-studied Approximate-Majority function, for example.

▷ **Claim 16.** Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a partial function defined as follows:

$$f(x) = \begin{cases} 0 & |x| \leq n/3 \\ 1 & |x| \geq 2n/3. \end{cases}$$

Then, $\text{rank}(f) = \Theta(n)$ and $\text{rank}(f) = \Theta(1)$.



■ **Figure 1** Complete AND-OR tree of depth 4.

Proof. Clearly $\text{rank}(f) = O(n)$. For the lower bound, we use the equivalence from Claim 15. A valid Delayer strategy is as follows: allow the Prover to choose input values for their first $n/3$ queries. It is easy to see that no matter what values the Prover chooses, the function can never be restricted to become a constant after these $n/3$ queries. Thus, $\text{val}(f) \geq n/3$ (and this can be easily seen to be tight). The randomized rank upper bound follows from the easy fact that $R(f) = O(1)$ and $\text{rrank}(f) \leq R(f)$ for all f . \triangleleft

When we restrict f to be a total function, it is no longer clear whether or not randomized rank can be significantly smaller than rank. In view of the example above, one might be tempted to consider functions that witness maximal separations between deterministic and randomized query complexity. The current state-of-the-art separation of $D(f) = \Omega(n)$ vs. $R(f) = \tilde{O}(\sqrt{n})$ is witnessed by variants of “pointer jumping” functions [15, 1, 24]. One might hope to use a similar argument as in the proof of Claim 16 to show that $\text{rank}(f) = \Omega(n)$ (and a randomized rank upper bound of $\tilde{O}(\sqrt{n})$ immediately follows from the randomized query upper bound). However, it is easy to show that the rank of these functions is actually $\tilde{O}(\sqrt{n})$, rendering this approach useless for these variants of pointer jumping functions. Nevertheless, we are able to use such an approach to show a separation between rank and randomized rank for another function whose deterministic and randomized query complexities are polynomially separated.

In the remainder of this section, let $F : \{0, 1\}^n \rightarrow \{0, 1\}$ be defined as the function evaluated by a complete $(\log n)$ -depth binary tree as described below. Assume $\log n$ to be an even integer, and the top node of this tree to be an OR gate. Nodes in subsequent layers alternate between AND’s and OR’s, and nodes at the bottom layer contain variables. We call this the complete AND-OR tree of depth $\log n$. See Figure 1 for a depiction of the complete depth-4 AND-OR tree.

It is easy to see via an adversarial argument that $D(F) = n$. Saks and Wigderson [31] showed that $R(F) = \Theta(n^{\log \frac{1+\sqrt{33}}{4}}) \approx \Theta(n^{0.753\dots})$.

► **Theorem 17** ([31, Theorem 1.5]). *Let $F : \{0, 1\}^n \rightarrow \{0, 1\}$ be the complete AND-OR tree of depth $\log n$. Then*

$$D(F) = n, \quad R(F) = \Theta\left(n^{\log \frac{1+\sqrt{33}}{4}}\right) \approx \Theta(n^{0.753\dots}).$$

We show that the rank of F equals $(n + 2)/3$.

15:10 Improved Quantum Query Upper Bounds Based on Classical Decision Trees

► **Theorem 18.** *Let $F : \{0, 1\}^n \rightarrow \{0, 1\}$ be the complete AND-OR tree of depth $\log n$. Then*

$$\text{rank}(F) = \frac{n+2}{3}.$$

For a proof we refer the reader to the full version [12, Proof of Theorem 3.9]. Since randomized rank is at most randomized decision tree complexity, F witnesses a separation between rank and randomized rank. This proves Theorem 6.

3 Proof of Theorem 8

In order to give a proof of Theorem 8, we require some useful properties of the non-binary span program with orthogonal inputs (NBSPwOI) and dual adversary solution constructed by Beigi and Taghavi [5]. For details of these constructions, we refer the reader to Appendix B.

The main result of Beigi and Taghavi [4] that we need is stated in the theorem below.

► **Theorem 19** ([4]). *Let $f : D_f \rightarrow [m]$ be a function with $D_f \subseteq [\ell]^n$, and let (P, w, \bar{w}) be a NBSPwOI computing f . Then,*

$$Q(f) = O(\text{wsize}(P, w, \bar{w})).$$

The main results of this section provide a characterization of the optimal witness complexity and objective value of the dual adversary bound, based on the weighting scheme in Beigi and Taghavi's construction, in terms of the objective value of Program 1.

► **Theorem 20.** *Let $f \subseteq \{0, 1\}^n \times \mathcal{R}$ be a relation, let \mathcal{T} be a decision tree computing f and let $\text{OPT}_{\mathcal{T}}$ denote the optimal value of Program 1. Then, for the construction of (P, w, \bar{w}) with variable weights as in Appendix B,*

$$\text{wsize}(P, w, \bar{w}) \leq \text{OPT}_{\mathcal{T}}.$$

Similarly, it is known that solutions to the dual adversary program (Program 2) yield quantum query upper bounds.

► **Theorem 21** ([22]). *Let $f : D_f \rightarrow [m]$ be a function with $D_f \subseteq [\ell]^n$, let C denote the optimal value of Program 2 for f . Then*

$$Q(f) = O(C). \tag{1}$$

► **Theorem 22.** *Let $f \subseteq \{0, 1\}^n \times \mathcal{R}$ be a relation, and let \mathcal{T} be a decision tree computing it. Let C denote the optimal value of Program 2 with variable weights as in Appendix B, and let $\text{OPT}_{\mathcal{T}}$ denote the optimal value of Program 1. Then $C = \text{OPT}_{\mathcal{T}}$.*

We now prove Theorem 8, using these results.

Proof of Theorem 8. We give two proofs, one via span program witness complexity, and another via the dual adversary bound.

1. Consider the NBSPwOI (P, w, \bar{w}) for \tilde{f} as constructed in Sections B.1 and B.2. Theorem 20 implies $\text{wsize}(P, w, \bar{w}) \leq \text{OPT}_{\mathcal{T}}$. Theorem 19 implies $Q(\tilde{f}) = O(\text{OPT}_{\mathcal{T}})$.
2. Consider the dual adversary solution for \tilde{f} as constructed in Appendix B.3. Theorems 21 and 22 imply $Q(\tilde{f}) = O(C) = O(\text{OPT}_{\mathcal{T}})$. ◀

4 An Optimal Weight Assignment

It now remains to investigate how we can assign weights to edges in a decision tree so as to optimize the objective value of Program 1. Beigi and Taghavi gave an explicit weighting scheme by coloring the edges decision tree with two colors, and then assigning weights depending on the color that the edge is colored by [5, Section 3]. They raise the question whether their scheme can be significantly improved upon. We answer this question affirmatively, by giving an *optimal* solution to the weight optimization program from Definition 7, and providing a constructive algorithm to compute the optimal weights in this section. We also give an alternative, albeit sub-optimal, assignment of weights in the full version of this paper [12, Appendix B].

The construction we present is recursive, in the sense that we first assign weights to the edges connected to the leaves, and subsequently work our way up the tree until we reach the root node. More precisely, in the i 'th iteration, we assign weights to all edges that have maximum distance i from a leaf.

First, we define the weight assignment, which we will refer to as the *canonical weight assignment*, in Definition 23. Then, we prove its optimality, resulting in Theorem 26. Finally, we prove Corollary 9, which gives some upper bounds on the optimal objective value, in terms of natural measures of the decision tree.

► **Definition 23** (Canonical weight assignment). *Let \mathcal{T} be a non-trivial decision tree with root node r . Let L and R be the two children nodes of r , connected to r by the edges e_L and e_R , respectively. Let \mathcal{T}_L and \mathcal{T}_R be the subtrees of \mathcal{T} rooted at L and R , respectively. Then, we assign weights to e_L and e_R , by setting*

$$W_{e_L} = \frac{\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2},$$

$$W_{e_R} = \frac{\text{OPT}_{\mathcal{T}_R} - \text{OPT}_{\mathcal{T}_L} + \sqrt{(\text{OPT}_{\mathcal{T}_R} - \text{OPT}_{\mathcal{T}_L})^2 + 4}}{2}.$$

In order to define the weights W_{e_L} and W_{e_R} , we need to know the optimal values of Program 1 for the subtrees \mathcal{T}_L and \mathcal{T}_R . We now proceed to show how one can compute these optimal values via a recurrence relation.

► **Lemma 24.** *Let \mathcal{T} be a non-trivial decision tree, and let L and R be the two children nodes of the root node. Let \mathcal{T}_L and \mathcal{T}_R be the subtrees of \mathcal{T} rooted at L and R , respectively. Then,*

$$\text{OPT}_{\mathcal{T}} \leq \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2}.$$

Proof. Suppose we have weight assignments W_L and W_R on the subtrees \mathcal{T}_L and \mathcal{T}_R , and positive parameters α_L, α_R and β_L, β_R such that they form feasible solutions to the optimization programs for the left and right subtrees \mathcal{T}_L and \mathcal{T}_R , and such that they attain the optimal values $\sqrt{\alpha_L \beta_L} = \text{OPT}_{\mathcal{T}_L}$ and $\sqrt{\alpha_R \beta_R} = \text{OPT}_{\mathcal{T}_R}$. Now, let the weighting scheme W on \mathcal{T} be defined such that $W_e = \sqrt{\beta_L / \alpha_L} (W_L)_e$ for all edges e in \mathcal{T}_L and $W_{e'} = \sqrt{\beta_R / \alpha_R} (W_R)_{e'}$ for all edges e' in \mathcal{T}_R , and choose W_{e_L} and W_{e_R} as in Definition 23. Furthermore, let

$$\alpha := \max \{ \text{OPT}_{\mathcal{T}_L} + W_{e_R}, \text{OPT}_{\mathcal{T}_R} + W_{e_L} \}, \quad \beta := \max \left\{ \text{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}}, \text{OPT}_{\mathcal{T}_R} + \frac{1}{W_{e_R}} \right\}. \quad (2)$$

15:12 Improved Quantum Query Upper Bounds Based on Classical Decision Trees

For every path $P \in P(\mathcal{T})$ containing e_L (essentially the same calculation also shows the same upper bound for paths containing e_R), we have

$$\sum_{e \in \bar{P}} \frac{1}{W_e} = \frac{1}{W_{e_L}} + \sqrt{\frac{\alpha_L}{\beta_L}} \sum_{e \in P \setminus \{e_L\}} \frac{1}{(W_L)_e} \leq \sqrt{\alpha_L \beta_L} + \frac{1}{W_{e_L}} = \text{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}} \leq \beta,$$

For every path $P \in P(\mathcal{T})$ containing e_L (essentially the same calculation also shows the same upper bound for paths containing e_R), we have

$$\sum_{e \in \bar{P}} W_e = \sqrt{\frac{\beta_L}{\alpha_L}} \sum_{e \in \bar{P} \setminus \{e_R\}} (W_L)_e + W_{e_R} \leq \sqrt{\alpha_L \beta_L} + W_{e_R} = \text{OPT}_{\mathcal{T}_L} + W_{e_R} \leq \alpha,$$

Thus all constraints of Program 1 for \mathcal{T} are satisfied with these values of α, β and the weighting scheme W . Hence,

$$\text{OPT}_{\mathcal{T}} \leq \sqrt{\alpha\beta} = \sqrt{\max\left\{\text{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}}, \text{OPT}_{\mathcal{T}_R} + \frac{1}{W_{e_R}}\right\} \cdot \max\{\text{OPT}_{\mathcal{T}_L} + W_{e_R}, \text{OPT}_{\mathcal{T}_R} + W_{e_L}\}}. \quad (3)$$

Finally, observe from our choices of W_{e_L} and W_{e_R} from Definition 23 that

$$\begin{aligned} \frac{1}{W_{e_L}} &= \frac{2}{\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}} \\ &= \frac{2(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R} - \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4})}{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 - (\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 - 4} \\ &= \frac{\text{OPT}_{\mathcal{T}_R} - \text{OPT}_{\mathcal{T}_L} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2} = W_{e_R}. \end{aligned}$$

Hence Equation (3) implies

$$\text{OPT}_{\mathcal{T}} \leq \max\{\text{OPT}_{\mathcal{T}_L} + W_{e_R}, \text{OPT}_{\mathcal{T}_R} + W_{e_L}\} = \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2},$$

where the last equality follows from our choices of W_{e_L} and W_{e_R} from Definition 23. This completes the proof. \blacktriangleleft

We next show that this weight assignment is optimal.

► **Lemma 25.** *Let \mathcal{T} be a non-trivial decision tree, and let L and R be the two children nodes of the root node. Let \mathcal{T}_L and \mathcal{T}_R be the subtrees of \mathcal{T} rooted at L and R , respectively. Then,*

$$\text{OPT}_{\mathcal{T}} \geq \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2}.$$

Proof. Suppose we have a weight assignment W and variables $\alpha, \beta > 0$, such that W , α and β form a feasible solution to the optimization program on \mathcal{T} , and attain optimality. We define

$$\begin{aligned} \alpha_L &= \max_{P \in P(\mathcal{T}_L)} \sum_{e \in \bar{P}} W_e, & \beta_L &= \max_{P \in P(\mathcal{T}_L)} \sum_{e \in P} \frac{1}{W_e}, \\ \alpha_R &= \max_{P \in P(\mathcal{T}_R)} \sum_{e \in \bar{P}} W_e, & \beta_R &= \max_{P \in P(\mathcal{T}_R)} \sum_{e \in P} \frac{1}{W_e}. \end{aligned}$$

Observe that $W|_{\mathcal{T}_L}$, α_L and β_L give a feasible solution to Program 1 for \mathcal{T}_L . Similarly, $W|_{\mathcal{T}_R}$, α_R and β_R give a feasible solution to Program 1 for \mathcal{T}_R . This implies that

$$\sqrt{\alpha_L \beta_L} \geq \text{OPT}_{\mathcal{T}_L}, \quad \sqrt{\alpha_R \beta_R} \geq \text{OPT}_{\mathcal{T}_R}. \quad (4)$$

Furthermore, we have

$$\begin{aligned}\alpha &= \max_{P \in P(\mathcal{T})} \sum_{e \in \bar{P}} W_e = \max \left\{ \max_{P \in P(\mathcal{T}_L)} \sum_{e \in \bar{P}} W_e + W_{e_R}, \max_{P \in P(\mathcal{T}_R)} \sum_{e \in \bar{P}} W_e + W_{e_L} \right\} \\ &= \max \{ \alpha_L + W_{e_R}, \alpha_R + W_{e_L} \},\end{aligned}\tag{5}$$

and similarly

$$\begin{aligned}\beta &= \max_{P \in P(\mathcal{T})} \sum_{e \in P} \frac{1}{W_e} = \max \left\{ \max_{P \in P(\mathcal{T}_L)} \sum_{e \in P} \frac{1}{W_e} + \frac{1}{W_{e_L}}, \max_{P \in P(\mathcal{T}_R)} \sum_{e \in P} \frac{1}{W_e} + \frac{1}{W_{e_R}} \right\} \\ &= \max \left\{ \beta_L + \frac{1}{W_{e_L}}, \beta_R + \frac{1}{W_{e_R}} \right\}.\end{aligned}\tag{6}$$

Finally, let $\gamma = \sqrt{\beta/\alpha}$, and observe that

$$\text{OPT}_{\mathcal{T}} = \sqrt{\alpha\beta} = \frac{\beta}{\gamma} = \gamma\alpha = \frac{1}{2} \left[\frac{\beta}{\gamma} + \gamma\alpha \right].\tag{7}$$

Now, let

$$\delta = \frac{1}{2} + \frac{\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R}}{2\sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}},\tag{8}$$

and observe that $\delta \in [0, 1]$. Thus,

$$\begin{aligned}\text{OPT}_{\mathcal{T}} &= \frac{1}{2} \left[\frac{\beta}{\gamma} + \gamma\alpha \right] && \text{by Equation (7)} \\ &= \frac{1}{2} \left[\max \left\{ \frac{\beta_L}{\gamma} + \frac{1}{\gamma W_{e_L}}, \frac{\beta_R}{\gamma} + \frac{1}{\gamma W_{e_R}} \right\} + \max \{ \gamma\alpha_L + \gamma W_{e_R}, \gamma\alpha_R + \gamma W_{e_L} \} \right] \\ &&& \text{by Equations (5) and (6)} \\ &\geq \frac{1}{2} \left[\delta \left(\frac{\beta_L}{\gamma} + \frac{1}{\gamma W_{e_L}} \right) + (1-\delta) \left(\frac{\beta_R}{\gamma} + \frac{1}{\gamma W_{e_R}} \right) + \delta(\gamma\alpha_L + \gamma W_{e_R}) + (1-\delta)(\gamma\alpha_R + \gamma W_{e_L}) \right] \\ &&& \text{since } \max\{a, b\} \geq \delta a + (1-\delta)b \text{ as } \delta \in [0, 1] \\ &= \frac{1}{2} \left[\delta \left(\frac{\beta_L}{\gamma} + \gamma\alpha_L \right) + (1-\delta) \left(\frac{\beta_R}{\gamma} + \gamma\alpha_R \right) + \left(\frac{\delta}{\gamma W_{e_L}} + (1-\delta)\gamma W_{e_L} \right) + \left(\delta\gamma W_{e_R} + \frac{1-\delta}{\gamma W_{e_R}} \right) \right] \\ &&& \text{rearranging terms} \\ &\geq \delta\sqrt{\alpha_L\beta_L} + (1-\delta)\sqrt{\alpha_R\beta_R} + \sqrt{\delta(1-\delta)} + \sqrt{\delta(1-\delta)} && \text{since } (a+b)/2 \geq \sqrt{ab} \text{ for all } a, b \geq 0 \\ &\geq \delta\text{OPT}_{\mathcal{T}_L} + (1-\delta)\text{OPT}_{\mathcal{T}_R} + 2\sqrt{\delta(1-\delta)} && \text{by Equation (4)} \\ &= \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R}}{2} + \frac{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2}{2\sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}} + 2\sqrt{\frac{1}{4} - \frac{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2}{4((\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4)}} \\ &&& \text{plugging the value of } \delta \text{ from Equation (8)} \\ &= \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R}}{2} + \frac{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2}{2\sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}} + \sqrt{\frac{4}{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}} \\ &= \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R}}{2} + \frac{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}{2\sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}} \\ &= \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2},\end{aligned}$$

completing the proof. \blacktriangleleft

We can now combine both lemmas above into the following theorem, providing a recursive characterization of the optimal value of Program 1 for all decision trees.

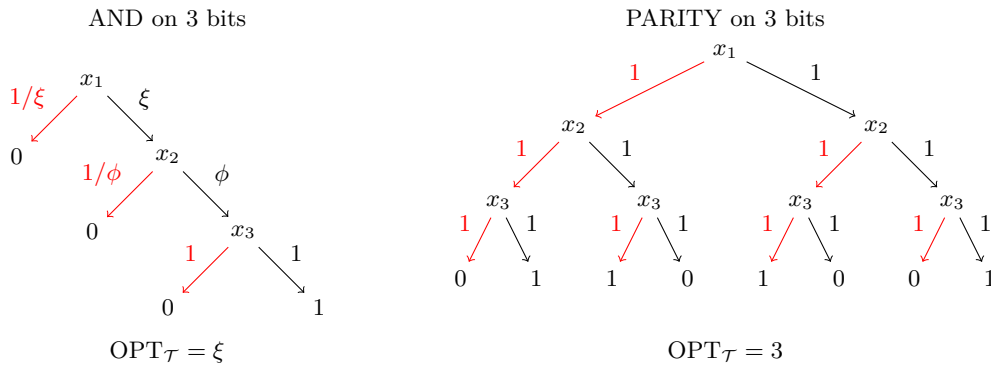
15:14 Improved Quantum Query Upper Bounds Based on Classical Decision Trees

► **Theorem 26.** Let \mathcal{T} be a non-trivial decision tree, and let L and R be the two children nodes of the root node. Let \mathcal{T}_L and \mathcal{T}_R be the subtrees of \mathcal{T} rooted at L and R , connected to the root node by edges e_L and e_R , respectively. Then,

$$\text{OPT}_{\mathcal{T}} = \frac{\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R} + \sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{2}.$$

Proof. The upper bound on $\text{OPT}_{\mathcal{T}}$ follows from Lemma 24 and the lower bound follows from Lemma 25. ◀

Observe that Lemma 24 can be used to recursively assign optimal weights to the edges of any given decision tree. We display this technique in two examples in Figure 2. Note that the objective value of Program 1 for a trivial decision tree (a single node) is 0, which provides the basis for our recursion.



■ **Figure 2** Examples of optimal weight assignments for two different decision trees. The red and black edges indicate the edges taken when the output of the query is 0 and 1, respectively, and the edge labels represent the weights. Left: Canonical weight assignment of the decision tree for the AND function on 3 bits, where $\phi = \frac{1+\sqrt{5}}{2}$, and $\xi = \frac{\phi+\sqrt{\phi+5}}{2}$. The objective value is ξ . Right: Canonical weight assignment of the decision tree for the PARITY on 3 bits, with optimal value 3.

Next, there are several ways in which we can conveniently upper bound the optimum value of Program 1 in terms of well-studied measures of the underlying decision tree. Corollary 9 exhibits two such bounds.

Proof of Corollary 9. Theorem 8 implies that it suffices to prove both of the required upper bounds on $\text{OPT}_{\mathcal{T}}$ rather than $Q(f)$. The rank-depth bound follows directly from the bound $\text{OPT}_{\mathcal{T}} \leq 2\sqrt{G(\mathcal{T}) \cdot \text{depth}(\mathcal{T})}$ derived in [5, Theorem 2], and the equality $G(\mathcal{T}) = \text{rank}(\mathcal{T})$ from Claim 12.

For proving the size bound, we use induction to show that $\text{OPT}_{\mathcal{T}} \leq \sqrt{2\text{DTSize}(\mathcal{T})}$. First observe that it is true for the trivial decision tree. Next, suppose that it is true for the left and right subtrees \mathcal{T}_L and \mathcal{T}_R of \mathcal{T} . That is,

$$\text{OPT}_{\mathcal{T}_L} \leq \sqrt{2\text{DTSize}(\mathcal{T}_L)}, \quad \text{and} \quad \text{OPT}_{\mathcal{T}_R} \leq \sqrt{2\text{DTSize}(\mathcal{T}_R)}.$$

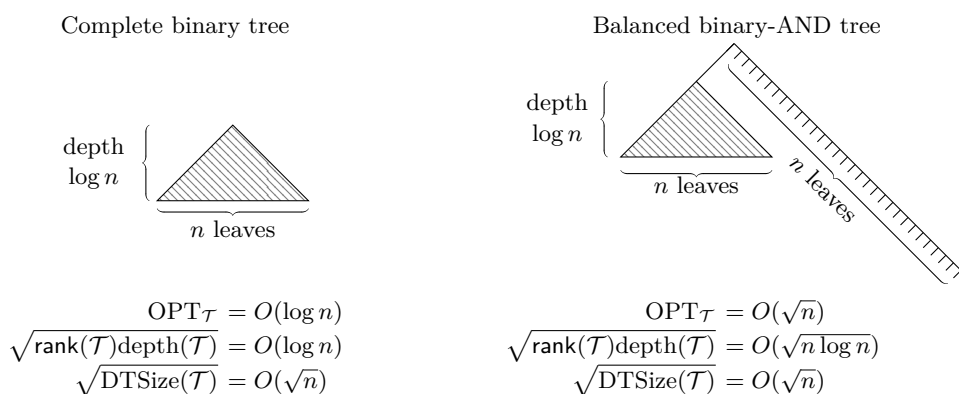
Then, by Theorem 26, the square of the optimal value of Program 1 for \mathcal{T} equals

$$\begin{aligned} \text{OPT}_{\mathcal{T}}^2 &= \frac{(\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R})^2 + (\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}{4} \\ &\quad + \frac{2(\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R})\sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{4} \end{aligned}$$

$$\begin{aligned}
 &= \frac{2(\text{OPT}_{\mathcal{T}_L}^2 + \text{OPT}_{\mathcal{T}_R}^2) + 4}{4} + \frac{2(\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R})\sqrt{(\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}}{4} \\
 &\leq \frac{2(\text{OPT}_{\mathcal{T}_L}^2 + \text{OPT}_{\mathcal{T}_R}^2) + 4 + (\text{OPT}_{\mathcal{T}_L} + \text{OPT}_{\mathcal{T}_R})^2 + (\text{OPT}_{\mathcal{T}_L} - \text{OPT}_{\mathcal{T}_R})^2 + 4}{4} \\
 &\hspace{15em} \text{since } 2ab \leq a^2 + b^2 \text{ for all } a, b \geq 0 \\
 &= \text{OPT}_{\mathcal{T}_L}^2 + \text{OPT}_{\mathcal{T}_R}^2 + 2 \\
 &\leq 2\text{DTSize}(\mathcal{T}_L) + 2\text{DTSize}(\mathcal{T}_R) + 2 = 2\text{DTSize}(\mathcal{T}).
 \end{aligned}$$

This completes the proof. ◀

Next, we note that the rank-depth and size bounds from Corollary 9 are incomparable, as witnessed by the examples displayed in Figure 3. In particular, our bounds are strictly stronger than those given by earlier works (Theorem 1) for the second tree in the figure.



■ **Figure 3** Examples showing separations between the two bounds derived in Corollary 9. The shaded regions represent complete binary trees. In the left example the rank-depth bound beats the size bound, whereas in the right example the opposite is true.

Finally, we note that we can obtain analogous quantum query upper bounds to those in Corollary 9 when the initial tree is a randomized one.

► **Corollary 27.** *Let $f \subseteq \{0, 1\}^n \times \mathcal{R}$ be a relation. Then $Q_{2/5}(f) = O(\sqrt{\text{RDTSize}(f)})$. Moreover, let \mathcal{T} be a randomized decision tree computing f with depth T . Then $Q_{2/5}(f) = O(\sqrt{\text{rrank}(\mathcal{T})T})$.*

The proof of Corollary 27 follows along similar lines as the proof of Theorem 5 and we omit it.

References

- 1 Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *Journal of the ACM*, 64(5):32:1–32:24, 2017. Earlier version in STOC’16. doi:10.1145/3106234.
- 2 Andris Ambainis, Aleksandrs Belovs, Oded Regev, and Ronald de Wolf. Efficient quantum algorithms for (gapped) group testing and junta testing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 903–922. SIAM, 2016. doi:10.1137/1.9781611974331.ch65.

- 3 Agnis Arins. Span-program-based quantum algorithms for graph bipartiteness and connectivity. In *Mathematical and Engineering Methods in Computer Science – 10th International Doctoral Workshop, MEMICS, Selected Papers*, volume 9548 of *Lecture Notes in Computer Science*, pages 35–41. Springer, 2015. doi:10.1007/978-3-319-29817-7_4.
- 4 Salman Beigi and Leila Taghavi. Span program for non-binary functions. *Quantum Information and Computation*, 19(9&10):760–792, 2019. doi:10.26421/QIC19.9-10-2.
- 5 Salman Beigi and Leila Taghavi. Quantum speedup based on classical decision trees. *Quantum*, 4:241, 2020. doi:10.22331/q-2020-03-02-241.
- 6 Salman Beigi, Leila Taghavi, and Artin Tajdini. Time and query optimal quantum algorithms based on decision trees. *CoRR*, abs/2105.08309, 2021. arXiv:2105.08309.
- 7 Aleksandrs Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 207–216. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.18.
- 8 Aleksandrs Belovs. Quantum dual adversary for hidden subgroups and beyond. In *Proceedings of the 18th International Conference on Unconventional Computation and Natural Computation (UCNC)*, volume 11493 of *Lecture Notes in Computer Science*, pages 30–36. Springer, 2019. doi:10.1007/978-3-030-19311-9_4.
- 9 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, volume 7501 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 2012. doi:10.1007/978-3-642-33090-2_18.
- 10 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.
- 11 Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Information and Computation*, 18(1&2):18–50, 2018. doi:10.26421/QIC18.1-2-2.
- 12 Arjan Cornelissen, Nikhil S. Mande, and Subhasree Patro. Improved quantum query upper bounds based on classical decision trees, 2022. doi:10.48550/ARXIV.2203.02968.
- 13 Yogesh Dahiya and Meena Mahajan. On (simple) decision tree rank. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 213 of *LIPICs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.FSTTCS.2021.15.
- 14 Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989. doi:10.1016/0890-5401(89)90001-1.
- 15 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM Journal on Computing*, 47(6):2435–2450, 2018. Earlier version in FOCS’15. doi:10.1137/16M1059369.
- 16 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219. ACM, 1996. doi:10.1145/237814.237866.
- 17 Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 526–535. ACM, 2007. doi:10.1145/1250790.1250867.
- 18 Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithms for connectivity and related problems. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *LIPICs*, pages 49:1–49:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.49.
- 19 Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017.

- 20 Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 102–111. IEEE Computer Society, 1993. doi:10.1109/SCT.1993.336536.
- 21 Robin Kothari. An optimal quantum algorithm for the oracle identification problem. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 25 of *LIPICs*, pages 482–493. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.STACS.2014.482.
- 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 Cedric Yen-Yu Lin and Han-Hsuan Lin. Upper bounds on quantum query complexity inspired by the Elitzur–Vaidman bomb tester. *Theory of Computing*, 12(1):1–35, 2016. doi:10.4086/toc.2016.v012a018.
- 24 Sagnik Mukhopadhyay, Jaikumar Radhakrishnan, and Swagato Sanyal. Separation between deterministic and randomized query complexity. *SIAM Journal on Computing*, 47(4):1644–1666, 2018. Earlier versions in FSTTCS’15 and FSTTCS’16. doi:10.1137/17M1124115.
- 25 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. URL: <https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition?format=HB>.
- 26 Noam Nisan. CREW prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. Earlier version in STOC’89. doi:10.1137/0220062.
- 27 Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k -sat (preliminary version). In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 128–136. ACM/SIAM, 2000. URL: <http://dl.acm.org/citation.cfm?id=338219.338244>.
- 28 Ben Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 544–551, 2009.
- 29 Ben Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–569. SIAM, 2011. doi:10.1137/1.9781611973082.44.
- 30 Ben Reichardt and Robert Špalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8(1):291–319, 2012. Earlier version in STOC’08. doi:10.4086/toc.2012.v008a013.
- 31 Michael E. Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 29–38. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.44.
- 32 Leila Taghavi. Simplified quantum algorithm for the oracle identification problem. *CoRR*, abs/2109.03902, 2021. arXiv:2109.03902.
- 33 Ronald de Wolf. Quantum computing: Lecture notes. *CoRR*, abs/1907.09415, 2019. arXiv:1907.09415.

A Preliminaries

All logarithms in this paper are taken base 2. For a bit $b \in \{0, 1\}$, let $\neg b$ denote the bit $1 - b$. Throughout this paper, \mathcal{R} denotes an arbitrarily large but finite set. For a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$, define the domain of f to be $D_f := \{x \in \{0, 1\}^n : \exists b \in \mathcal{R} \text{ such that } (x, b) \in f\}$. For a vector v , let $\|v\|$ denote its ℓ_2 -norm. For a matrix M , let $\|M\|$ denote its spectral

norm. For matrices M and N of the same dimensions, let $M \circ N$ denote their entry-wise (Hadamard) product. Let $\delta_{a,b}$ be the function that outputs 1 when $a = b$ and 0 otherwise. We use $[T]$ to denote the set $\{1, \dots, T\}$ where $T \in \mathbb{Z}^+$.

A.1 Decision Trees

A decision tree computing a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ is a binary tree with leaf nodes labeled in \mathcal{R} , each internal node is labeled by a variable x_i and has two outgoing edges, labeled 0 and 1. On input $x \in \{0, 1\}^n$, the tree's computation proceeds by computing x_i as indicated by the node's label and following the edge indicated by the value of the computed variable. The output value at the leaf, say $b \in \mathcal{R}$, must be such that $(x, b) \in f$. Given a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ and a deterministic decision tree \mathcal{T} computing it, define the function \tilde{f} that takes an input $x \in \{0, 1\}^n$ and outputs the leaf of \mathcal{T} reached on input x . Let $V(\mathcal{T})$ denote the set of vertices in \mathcal{T} , and, we use $V_I(\mathcal{T})$ to denote the set of internal nodes of \mathcal{T} . We use the notation $J(v)$ to denote the variable queried at vertex v . For a vertex $v \in V_I(\mathcal{T})$ and $q \in \{0, 1\}$, let $N(v, q)$ denote the vertex that is the child of v along the edge that has label q . For neighbors v, w , let $e(v, w)$ denote the edge between them. For an input $x \in D_f$, let P_x denote the unique path in \mathcal{T} from its root to $\tilde{f}(x)$. For a path P in \mathcal{T} , we say an edge e *deviates from* P if exactly one vertex of e is in P . For a path P in \mathcal{T} , define $\bar{P} := \{e : e \text{ deviates from } P\}$. We let $P(\mathcal{T})$ denote the set of all paths from the root to a leaf in \mathcal{T} . We assume that decision trees computing relations f contain no extraneous leaves, i.e., for all leaves there is an input $x \in D_f$ that reaches that leaf. We also assume that for every path P in \mathcal{T} and index $i \in [n]$, the variable x_i is queried at most once on P .

The decision tree complexity (also called *deterministic query complexity*) of f , denoted $D(f)$, is defined as

$$D(f) := \min_{\mathcal{T}: \mathcal{T} \text{ is a DT computing } f} \text{depth}(\mathcal{T}).$$

Note that a deterministic decision tree in fact computes a function, since each input reaches exactly one leaf on the computation path of the tree. A randomized decision tree is a distribution over deterministic decision trees. We say a randomized decision tree computes a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ with error $1/3$ if for all $x \in \{0, 1\}^n$, the probability of outputting a $b \in \mathcal{R}$ such that $(x, b) \in f$ is at least $2/3$. The depth of a randomized decision tree is the maximum depth of a deterministic decision tree in its support. Define the randomized decision tree complexity (also called *randomized query complexity*) of f as

$$R(f) := \min_{\substack{\mathcal{T}: \mathcal{T} \text{ is a randomized DT} \\ \text{that computes } f \text{ to error } 1/3}} \text{depth}(\mathcal{T}).$$

Another measure of interest to us in this work is the *decision-tree size complexity* of f .

► **Definition 28** (Decision-tree size complexity). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function. Define the decision-tree size complexity of f , which we denote by $\text{DTSize}(f)$, as*

$$\text{DTSize}(f) := \min_{\mathcal{T}: \mathcal{T} \text{ computes } f} \text{DTSize}(\mathcal{T}),$$

where $\text{DTSize}(\mathcal{T})$ denotes the number of nodes of \mathcal{T} . Analogously, the randomized decision-tree size complexity of f is defined to be

$$\text{RDTSize}(f) := \min_{\substack{\mathcal{T}: \mathcal{T} \text{ is a randomized DT} \\ \text{that computes } f \text{ to error } 1/3}} \text{RDTSize}(\mathcal{T}),$$

where $\text{RDTSize}(\mathcal{T})$ denotes the maximum number of nodes of a decision tree in the support of \mathcal{T} .

It is easy to observe that the number of nodes in a deterministic decision tree equals one less than twice the number of leaves in the tree.

A.2 Quantum Query Complexity

We refer the reader to [25, 33] for the basics of quantum computing. A quantum query algorithm \mathcal{A} for a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ begins in a fixed an initial state $|\psi_0\rangle$, applies a sequence of unitaries $U_0, O_x, U_1, O_x, \dots, U_T$, and performs a measurement. Here, the initial state $|\psi_0\rangle$ and the unitaries U_0, U_1, \dots, U_T are independent of the input. The unitary O_x represents the “query” operation, and maps $|i\rangle|b\rangle$ to $|i\rangle|b + x_i \bmod 2\rangle$ for all $i \in [n]$ and $|0\rangle$ to $|0\rangle$. We say that \mathcal{A} is an ε -error algorithm computing f if for all x in the domain of f , the probability of outputting $b \in \mathcal{R}$ such that $(x, b) \in f$ is at least $1 - \varepsilon$. The ε -error quantum query complexity of f , denoted by $Q_\varepsilon(f)$, is the least number of queries required for a quantum query algorithm to compute f with error ε . When the subscript ε is dropped we assume $\varepsilon = 1/3$; the bounded-error query complexity of f is $Q(f)$.

B Construction of Span Programs and Dual Adversary Solution of [5]

In this section, we describe Beigi and Taghavi’s construction of an NBSPwOI and a dual adversary solution for \tilde{f} given a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ and a decision tree \mathcal{T} computing it [5, Section 3] (recall from Appendix A that \tilde{f} takes an input $x \in \{0, 1\}^n$ and outputs the leaf of \mathcal{T} reached on input x). We describe their construction in a modular fashion: we leave the choice of “weights” of the vectors in the span program and dual adversary solution unfixed. We show that the witness complexity and dual adversary bounds thus obtained are captured by the objective value of Program 1. In the next section we demonstrate a choice of weights and prove its optimality. We exhibit another interesting choice of weights in the full version of this paper [12, Appendix B].

B.1 Span Program Construction

The model of span programs of interest to us is that of “non-binary span programs with orthogonal inputs”, abbreviated NBSPwOI. This model was introduced by Beigi and Taghavi [4]. We refer the reader to the full version of this paper [12, Section 2.3] for basics.

In order to define the NBSPwOI, we first assign strictly positive real weights W_e to all edges e in the decision tree. These weights play a crucial role in the witness complexity analysis, presented in Appendix B.2.

The following is the NBSPwOI for \tilde{f} .

- The vector space is determined by the orthonormal basis indexed by vertices of \mathcal{T} : $\{|v\rangle : v \in V(\mathcal{T})\}$.
- The input vectors are

$$I_{j,q} = \bigcup_{v \in V_I(\mathcal{T}): J(v)=j} \left\{ \sqrt{W_{e(v, N(v,q))}} (|v\rangle - |N(v,q)\rangle) \right\}. \quad (9)$$

That is, for all $j \in [n]$ and $q \in \{0, 1\}$, the input vectors correspond to edges corresponding to answers of queries of the form $x_j = q$. In other words, for every vertex $v \in V_I(\mathcal{T})$, $e(v, N(v, x_{J(v)}))$ is always the unique available outgoing edge of v . Moreover, these vectors are weighted, and we leave these weights variable for now.

- Let r denote the root vertex of \mathcal{T} . For each leaf u of \mathcal{T} , the associated target vector is given by $|t_u\rangle = |r\rangle - |u\rangle$.

We now give positive and negative witnesses for every $x \in D_f$, argue that the above span program evaluates \tilde{f} , and analyze the positive and negative witness complexities.

B.2 Witness Complexity Analysis

Note that, we use $v \in P_x$ to denote a vertex in the path P_x , and, we use $e \in P_x$ to denote an edge in the path P_x . For every $x \in D_f$, we can express the corresponding target vector by a telescoping sum of vectors that are all available to x , as

$$\begin{aligned} |t_{\tilde{f}(x)}\rangle &= |r\rangle - |\tilde{f}(x)\rangle = \sum_{v \in P_x \setminus \{\tilde{f}(x)\}} |v\rangle - |N(v, x_{J(v)})\rangle \\ &= \sum_{v \in P_x \setminus \{\tilde{f}(x)\}} \frac{1}{\sqrt{W_{e(v, N(v, x_{J(v)})})}} \left(\sqrt{W_{e(v, N(v, x_{J(v)})})} (|v\rangle - |N(v, x_{J(v)})\rangle) \right). \end{aligned}$$

On the other hand, we let $|\bar{w}_x\rangle = \sum_{v \in P_x} |v\rangle$. For any vector in $|v'\rangle \in I(x)$, we have

$$\langle \bar{w}_x | v' \rangle = \sum_{v \in P_x} \sqrt{W_{e(v, N(v, x_{J(v)})})} (\langle v | v' \rangle - \langle N(v, x_{J(v)}) | v' \rangle) = 0.$$

For a leaf $u \neq \tilde{f}(x)$ of \mathcal{T} ,

$$\langle t_u | \bar{w}_x \rangle = \sum_{v \in P_x} \langle r | v \rangle - \sum_{v \in P_x} \langle u | v \rangle = 1 - 0 = 1.$$

This implies that the NBSwOI indeed computes \tilde{f} . For the positive and negative witness sizes, we have

$$\text{wsize}^+(P, w, \bar{w}) = \max_{x \in D_f} \sum_{v \in P_x \setminus \{\tilde{f}(x)\}} \frac{1}{W_{e(v, N(v, x_{J(v)})})} = \max_{P \in \mathcal{P}(\mathcal{T})} \sum_{e \in P} \frac{1}{W_e},$$

and

$$\begin{aligned} \text{wsize}^-(P, w, \bar{w}) &= \max_{x \in D_f} \|A^\dagger |\bar{w}_x\rangle\|^2 = \max_{x \in D_f} \left\| \sqrt{W_{e(v, N(v, x_{J(v)})})} (|v\rangle - \langle N(v, x_{J(v)}) | \bar{w}_x \rangle) \right\|^2 \\ &= \sum_{v \in P_x \setminus \{\tilde{f}(x)\}} W_{e(v, N(v, \neg x_{J(v)})}) = \max_{P \in \mathcal{P}(\mathcal{T})} \sum_{e \in \bar{P}} W_e. \end{aligned}$$

Now, it remains to find the weight assignment W that minimizes the total complexity of the NBSwOI, which is given by

$$\text{wsize}(P, w, \bar{w}) = \sqrt{\text{wsize}^-(P, w, \bar{w}) \cdot \text{wsize}^+(P, w, \bar{w})} \leq \sqrt{\max_{P \in \mathcal{P}(\mathcal{T})} \sum_{e \in P} \frac{1}{W_e} \cdot \max_{P \in \mathcal{P}(\mathcal{T})} \sum_{e \in \bar{P}} W_e}.$$

Thus, if we assign weights W to the edges of \mathcal{T} as in Sections B.1 and B.2, and set $\alpha = \max_{P \in \mathcal{P}(\mathcal{T})} \sum_{e \in \bar{P}} W_e$ and $\beta = \max_{P \in \mathcal{P}(\mathcal{T})} \sum_{e \in P} \frac{1}{W_e}$, the construction from earlier in this section gives rise to an explicit NBSwOI computing \tilde{f} with witness complexity of $\sqrt{\alpha\beta}$. This is exactly captured by Program 1, giving us Theorem 20.

In the next subsection we show that a solution to Program 1 also gives a dual adversary solution with the same objective value.

B.3 Dual Adversary Solution

We refer the reader to the full version of this paper [12, Section 2.4] for basics. Here in this section we give a simplified analysis of Beigi and Taghavi's construction of a dual adversary solution for a relation $f \subseteq \{0, 1\}^n \times \mathcal{R}$ given a deterministic decision tree \mathcal{T} that computes f . Recall that for a deterministic tree \mathcal{T} computing f , \tilde{f} is the function that takes input $x \in D_f$ and outputs the leaf of \mathcal{T} reached on input x . We show that a dual adversary solution for \tilde{f} can also be obtained by different settings of weights as in the previous subsection, and obtain a corresponding dual adversary bound as the optimum value of Program 1.

We construct vectors $\{|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{|w_{xj}\rangle : x \in D_f, j \in [n]\}$ that are feasible solutions to Program 2, which we recall below.

■ **Program 2** Dual SDP for \tilde{f} . By replacing the $\delta_{\tilde{f}(x), \tilde{f}(y)}$ term with $\delta_{f(x), f(y)}$ in the constraints we instead get the dual SDP for f .

Variables	$\{ u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{ w_{xj}\rangle : x \in D_f, j \in [n]\}, d$
Minimize	$\max_{x \in D_f} \max \left\{ \sum_{j=1}^n \ u_{xj}\rangle \ ^2, \sum_{j=1}^n \ w_{xj}\rangle \ ^2 \right\}$
s.t.	$\sum_{j \in [n]: x_j \neq y_j} \langle u_{xj} w_{yj} \rangle = 1 - \delta_{\tilde{f}(x), \tilde{f}(y)} \quad \forall x, y \in D_f$ $ u_{xj}\rangle, w_{xj}\rangle \in \mathbb{C}^d \quad \text{for all } x \in D_f$

Let $V_I(\mathcal{T})$ denote the set of internal nodes of \mathcal{T} . Consider the basis set $\{|v\rangle : v \in V_I(\mathcal{T})\}$. We construct the vectors $|u_{xj}\rangle$ and $|w_{xj}\rangle$ in the space $\mathbb{C}^{V_I(\mathcal{T})}$. Additionally, we use $V_j(\mathcal{T})$ to denote the set of vertices associated with query index j , i.e., $V_j(\mathcal{T}) = \{v \in V_I(\mathcal{T}) : J(v) = j\}$.

Define $|u_{xj}\rangle$ and $|w_{xj}\rangle$ as follows.

$$|u_{xj}\rangle = \begin{cases} \frac{1}{\sqrt{W_{e(v, N(v, x_j))}}} |v\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}), \\ 0 & \text{otherwise,} \end{cases} \quad (10)$$

and,

$$|w_{xj}\rangle = \begin{cases} \sqrt{W_{e(v, N(v, \neg x_j))}} |v\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}), \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

We claim that these vectors form a feasible solution to Program 2. Fix $x, y \in D_f$ with $\tilde{f}(x) \neq \tilde{f}(y)$. We now verify that the corresponding equality constraint in Program 2 is satisfied.

- There is a unique vertex in \mathcal{T} where P_x and P_y deviate. Let $v \in V_I(\mathcal{T})$ denote this vertex, and let its associated query index be $J(v) = i$. We have $v \in P_x \cap P_y$ and $x_i \neq y_i$. In that case $\langle u_{xi} | w_{yi} \rangle = 1$ by the definitions of $|u_{xi}\rangle$ and $|w_{yi}\rangle$ from Equations (10) and (11).
- Consider an index $j \in [n] \setminus \{i\}$ such that $x_j \neq y_j$. Let v' and v'' be the vertices on P_x and P_y , respectively (if they exist), with $J(v') = J(v'') = j$. By the previous point, $v' \notin P_y$ and $v'' \notin P_x$. Thus, $\langle v' | v'' \rangle = 0$ from Equations (10) and (11), which implies $\langle u_{xj} | w_{yj} \rangle = 0$.

Thus, we have for all $x, y \in D_f$ with $\tilde{f}(x) \neq \tilde{f}(y)$,

$$\sum_{j \in [n]: x_j \neq y_j} \langle u_{xj} | w_{yj} \rangle = 1 - \delta_{\tilde{f}(x), \tilde{f}(y)}.$$

In the case when $\tilde{f}(x) = \tilde{f}(y)$, the right hand side in the constraint evaluates to 0 and so does the left side, because the indices where x and y differ cannot be queried on their path since x and y reach the same leaf in \mathcal{T} . Therefore, the set of vectors $\{|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{|w_{xj}\rangle : x \in D_f, j \in [n]\}$, and $d = |V_I(\mathcal{T})|$ form a feasible solution to Program 2 for \tilde{f} .

15:22 Improved Quantum Query Upper Bounds Based on Classical Decision Trees

Proof of Theorem 22. Let C denote the objective value of Program 2 with the settings of $\{\|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{\|w_{xj}\rangle : x \in D_f, j \in [n]\}$ as defined in Equations (10) and (11), and $d = |V_I(\mathcal{T})|$.

We now argue that $C = \text{OPT}_{\mathcal{T}}$, where $\text{OPT}_{\mathcal{T}}$ denotes the optimal solution of Program 1. First note that for all $x \in D_f$,

$$\sum_{j=1}^n \|\|u_{xj}\rangle\|^2 = \sum_{e \in P_x} \frac{1}{W_e} \quad \text{and} \quad \sum_{j=1}^n \|\|w_{xj}\rangle\|^2 = \sum_{e \in \bar{P}_x} W_e.$$

Thus,

$$C = \min_{x \in D_f} \max \left\{ \sum_{j=1}^n \|\|u_{xj}\rangle\|^2, \sum_{j=1}^n \|\|w_{xj}\rangle\|^2 \right\} = \min_{x \in D_f} \max \left\{ \sum_{e \in P_x} \frac{1}{W_e}, \sum_{e \in \bar{P}_x} W_e \right\}.$$

Thus we can alternatively view C to be an optimal solution to the following optimization program.

■ **Program 3** Optimization program with C being its optimal solution.

Variables	$\{W_e : e \text{ is an edge in } \mathcal{T}\}, \alpha, \beta$		
Minimize	$\max\{\alpha, \beta\}$		
s.t.	$\sum_{e \in \bar{P}} W_e$	$\leq \alpha,$	for all paths $P \in P(\mathcal{T})$
	$\sum_{e \in P} \frac{1}{W_e}$	$\leq \beta,$	for all paths $P \in P(\mathcal{T})$
	W_e	$> 0,$	for all edges e in \mathcal{T}
	α, β	$> 0.$	

We now show $C = \text{OPT}_{\mathcal{T}}$. Let $\{W_e : e \text{ edge in } \mathcal{T}\}, \alpha, \beta$ be settings of variables in a feasible solution to Program 3, with objective value C . Clearly the same settings of variables also form a feasible solution to Program 1, since the constraints are the same. The corresponding objective value of Program 1 is $\sqrt{\alpha\beta} \leq \max\{\alpha, \beta\} = C$. Thus, $\text{OPT}_{\mathcal{T}} \leq C$.

In the other direction, let $\{W_e : e \text{ edge in } \mathcal{T}\}, \alpha, \beta$ be settings of variables in a feasible solution to Program 1 with objective value $\text{OPT}_{\mathcal{T}}$. Set

$$\begin{aligned} W'_e &= \sqrt{\beta/\alpha} \cdot W_e && \text{for all edges } e \text{ in } \mathcal{T}, \\ \alpha' &= \sqrt{\alpha\beta}, \\ \beta' &= \sqrt{\alpha\beta}. \end{aligned}$$

It is easy to verify that this setting of variables is feasible for Program 3, and attains objective value $\max\{\alpha', \beta'\} = \sqrt{\alpha\beta} = \text{OPT}_{\mathcal{T}}$. Thus, $C \leq \text{OPT}_{\mathcal{T}}$, proving the claim. ◀

On the VNP-Hardness of Some Monomial Symmetric Polynomials

Radu Curticapean  

IT University of Copenhagen, Denmark
Basic Algorithms Research Copenhagen, Denmark

Nutan Limaye  

IT University of Copenhagen, Denmark
Basic Algorithms Research Copenhagen, Denmark

Srikanth Srinivasan  

Department of Computer Science, Aarhus University, Denmark
On leave from Department of Mathematics, IIT Bombay, India

Abstract

A polynomial $P \in \mathbb{F}[x_1, \dots, x_n]$ is said to be symmetric if it is invariant under any permutation of its input variables. The study of symmetric polynomials is a classical topic in mathematics, specifically in algebraic combinatorics and representation theory. More recently, they have been studied in several works in computer science, especially in algebraic complexity theory.

In this paper, we prove the computational hardness of one of the most basic kinds of symmetric polynomials: the *monomial symmetric polynomials*, which are obtained by summing all distinct permutations of a single monomial. This family of symmetric functions is a natural basis for the space of symmetric polynomials (over any field), and generalizes many well-studied families such as the elementary symmetric polynomials and the power-sum symmetric polynomials.

We show that certain families of monomial symmetric polynomials are *VNP-complete* with respect to oracle reductions. This stands in stark contrast to the case of elementary and power symmetric polynomials, both of which have constant-depth circuits of polynomial size.

2012 ACM Subject Classification Theory of computation \rightarrow Algebraic complexity theory; Computing methodologies \rightarrow Representation of polynomials

Keywords and phrases algebraic complexity, symmetric polynomial, permanent, Sidon set

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.16

Funding Basic Algorithms Research Copenhagen is supported by Villum Foundation grant 16582. *Srikanth Srinivasan*: Supported by start-up grant from Aarhus University.

1 Introduction

This paper considers the algebraic complexity of *symmetric polynomials*: a multivariate polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ is said to be symmetric if it is invariant under any permutation of its variables x_1, \dots, x_n . Standard examples of such polynomials include the *elementary symmetric polynomials* and the *power-sum symmetric polynomials*. The study of symmetric polynomials is a classical topic in mathematics, especially in algebraic combinatorics and representation theory (see, e.g. [18, 14]). In particular, standard bases of homogeneous symmetric polynomials of fixed degree d and the matrices of linear transformations that translate between these bases are studied. For many natural bases, the entries of these matrices encode interesting combinatorial and representation-theoretic quantities.

An important example of such a basis of n -variate symmetric polynomials is the family of *monomial symmetric polynomials*, which are considered in this paper. In the following, we say that a partition λ of an integer $d \in \mathbb{N}$ is a non-increasingly ordered tuple of positive numbers $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_r)$ summing to d , i.e. $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r$ and $\sum_i \lambda_i = d$. We write $\lambda \vdash d$



© Radu Curticapean, Nutan Limaye, and Srikanth Srinivasan;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 16; pp. 16:1–16:14



Leibniz International Proceedings in Informatics

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

to indicate this fact. The monomial symmetric polynomial m_λ is the polynomial obtained by summing all distinct monomials $y_1^{\lambda_1} \cdots y_r^{\lambda_r}$ that can be obtained by picking y_1, \dots, y_r out of x_1, \dots, x_n without repetitions. These generalize both the elementary symmetric polynomials (obtained by taking $r = d$ and all $\lambda_i = 1$) and the power symmetric polynomials (obtained by taking $r = 1$ and $\lambda_1 = d$). It is also easily seen that any symmetric polynomial is a unique linear combination of monomial symmetric polynomials.

In this paper, we study monomial symmetric polynomials from the perspective of algebraic complexity. The complexity of general symmetric polynomials has already been investigated in various works, as summarized below.

- Many results in algebraic complexity concern the computational complexity of the *elementary* symmetric polynomials. Non-trivial upper bounds for computing these polynomials have been shown in various models [13, 16, 8], starting with the work of Nisan and Wigderson [13]. In particular, the upper bound by Shpilka and Wigderson [16] played a crucial role in recent work that proved the first superpolynomial lower bounds for constant-depth algebraic circuits [10]. Lower bounds for computing elementary symmetric polynomials have also been shown [13, 16, 15, 8, 6].
- The algebraic complexity of various symmetric polynomials in the *monotone* setting has been investigated [5, 7]. Here, the underlying field is the reals and we do not allow any negative constants in the underlying computation. In particular, the result of Grigoriev and Koshevoy [7] implies an exponential lower bound on monotone algebraic circuits computing certain monotone symmetric polynomials. However, this does not imply lower bounds for general (non-monotone) algebraic circuits, which are the focus of this paper.
- The fundamental theorem of symmetric polynomials states that any symmetric polynomial $p(x_1, \dots, x_n)$ can be written uniquely as a polynomial f_{elem} in the elementary symmetric polynomials. A recent result of Bläser and Jindal [2] shows that, over fields of characteristic 0, the polynomials p and f_{elem} have roughly the same algebraic circuit complexity. This implies the hardness of p when f_{elem} is a known hard polynomial such as the permanent, but it might be non-trivial to understand the complexity of f_{elem} in general. A variant of [2] was proved in [4], which holds for more general models of algebraic computation, but it requires technical conditions on f_{elem} .
- Monomial symmetric polynomials appear naturally in the context of learning theory, e.g., when estimating properties of distributions. Here, the learning algorithm has access to samples from a discrete distribution and is required to estimate a symmetric property of the distribution, e.g., the entropy or support size. Acharya, Das, Orlitsky and Suresh [1] analyzed algorithms based on a particular estimator and showed their optimality in a variety of settings. This estimator seeks to optimize a given monomial symmetric polynomial over the space of probability distributions. The problem we study in this paper, that is, *evaluating* a monomial symmetric polynomial at a given input, intuitively appears to be an easier computational problem.

Many of the above works try to understand the algebraic complexity of various families of monomial symmetric polynomials. However, to the best of our knowledge, it was not known if there are families of monomial symmetric polynomials that are hard for general algebraic circuits. We prove that, indeed, polynomial-sized circuits for certain monomial symmetric polynomials m_λ would imply that VNP collapses to VP. More formally, we show that these monomial symmetric polynomials are VNP-hard under c -reductions; these reductions will be introduced in Section 2. (Containment in VNP is easily seen, so VNP-completeness follows.)

► **Theorem 1** (Main theorem). *Fix an algebraically closed field of characteristic 0 or $q \geq 3$. There are two polynomial functions $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and an explicit¹ sequence of partitions $\lambda_1, \lambda_2, \dots$ such that $\lambda_n \vdash r(n)$ for $n \in \mathbb{N}$ and the following holds: If the polynomials $m_{\lambda_n}(x_1, \dots, x_{s(n)})$ admit algebraic circuits of polynomial size, then so does the permanent.*

The permanent of order n is a polynomial in $x_{i,j}$ for $1 \leq i, j \leq n$ and can be seen as a sum over all perfect matchings in a complete bipartite graph with $n + n$ vertices and an edge of weight $x_{i,j}$ between the i -th left and the j -th right vertex. Each perfect matching is weighted by the product of the weights of all involved edges. The hypergraph permanent is defined analogously for k -uniform hypergraphs.

Over characteristic 0, the reduction by Bläser and Jindal [2], augmented by an observation due to Chaugule et al. [4], implies that to prove the theorem, it suffices to establish the hardness of the polynomial combination f_{pow} that expresses m_{λ} in terms of the power-sum symmetric polynomials. Towards this, we show that a particular sum-product f_{match} over perfect matchings can be extracted from f_{pow} . However, the weights of perfect matchings M in f_{match} do not necessarily correspond to those in the permanent: A priori, it may not be possible to recover the edges present in M from the weight of M in f_{match} . This property can however be ensured by choosing the parts in λ from a *Sidon set*, a notion from additive combinatorics. In a Sidon set, any pair of distinct numbers is uniquely identified by its sum. We can apply this to uniquely recover the edges present in a matching from their weight in f_{match} .

Over characteristic $q \geq 3$, the proof is similar, but more involved: First, we need to cast f_{pow} as a polynomial combination f_{elem} in the elementary symmetric polynomials in order to invoke a known reduction by Chaugule et al. [4] that applies to fields of characteristic q . In this form, it will however be less obvious how to extract a sum-product over perfect matchings. Focussing on the homogeneous component of minimum degree in f_{elem} and carefully choosing λ will eventually allow us to extract a $(q - 1)$ -uniform hypergraph permanent from f_{elem} . Here, we also crucially exploit the characteristic of the field, along with basic properties of the transformation that expresses power-sum symmetric polynomials in terms of the elementary symmetric polynomials.

2 Preliminaries

We use boldface notation \mathbf{x}, \mathbf{y} for vectors. Throughout, λ will denote a *partition*, i.e. a sequence of weakly decreasing positive integers $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r \geq 1$. Here, r is called the *number of parts* of λ .

Symmetric polynomials

In the following, let \mathbb{F} be any field and let $\mathbf{x} = (x_1, \dots, x_n)$. We say that $P(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$ is *symmetric* if it is invariant under all permutations of the underlying variables. Examples of symmetric polynomials include the following:

- The *elementary symmetric polynomials* $e_{n,d} = \sum_S \prod_{i \in S} x_i$ for $d \leq n$, where S ranges over all d -element subsets of $[n]$. If n is implicit from context, we set $e_d := e_{n,d}$.
- The *power-sum symmetric polynomials* $p_{n,d} = \sum_{i=1}^n x_i^d$. If n is implicit from context, we denote this polynomial by p_d .

¹ The sequence of partitions is explicit in the sense that there is a polynomial-time algorithm that computes λ_n on input 1^n .

16:4 On the VNP-Hardness of Some Monomial Symmetric Polynomials

- More generally, given a partition λ with $r \leq n$ parts, the *monomial symmetric polynomial* m_λ is the sum of all monomials where the distinct exponents are exactly $\lambda_1, \dots, \lambda_r$. In particular, when $\lambda_1, \dots, \lambda_r$ are all distinct, we can define this polynomial by

$$m_\lambda = \sum_{\substack{i_1, \dots, i_r \in [n] \\ \text{distinct}}} x_{i_1}^{\lambda_1} \cdots x_{i_r}^{\lambda_r}.$$

As noted in the introduction, the elementary and power-sum symmetric polynomials are special cases of monomial symmetric polynomials.

The following basic theorem regarding symmetric polynomials will be important.

- **Theorem 2** (Fundamental theorem of symmetric polynomials (see, e.g., [11])). *For any symmetric polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$, there is a unique polynomial $f_{\text{elem}}(y_1, \dots, y_n)$ with $f_{\text{elem}}(e_1, \dots, e_n) = f(\mathbf{x})$. If \mathbb{F} has characteristic zero, then there is also a unique polynomial $f_{\text{pow}}(y_1, \dots, y_n)$ that represents f analogously in terms of the power-sum symmetric polynomials.*

Further, both f_{elem} and f_{pow} (the latter over characteristic 0) have degree at most $\deg(f)$ and do not depend on y_i for $i > \deg(f)$.

Algebraic circuits and Oracle reductions

We work throughout with the standard algebraic circuit model. We refer the reader to standard resources [3, 17] for definitions and basic results regarding the model. We recall also the notion of *c-reductions* between two polynomials f and g : We define $L^g(f)$ to be the smallest s such that the polynomial f is computed by an algebraic circuit C of size at most s that is additionally allowed to use gates for the polynomial g . If $L^g(f)$ is bounded by a polynomial in the number of variables and degree of f and g , we also say that f admits a *c-reduction* to g and write $f \preceq_c g$.

A result of Bläser and Jindal [2] relates the algebraic complexity of a symmetric polynomial f with its associated polynomial f_{elem} , when the underlying field is the field of complex numbers. Chaugule et al. [4, Theorem 4.16] extended the result to f_{pow} .

- **Theorem 3** ([2, 4]). *Any symmetric polynomial $f \in \mathbb{C}[\mathbf{x}]$ admits the reductions $f_{\text{elem}} \preceq_c f$ and $f_{\text{pow}} \preceq_c f$.*

We also need the following variant of Theorem 3 due to [4]. While the results of [4] are stated for characteristic zero, we show in Section 5 how to modify them to work for positive characteristic in the setting we are interested in.

In the following, given a polynomial $f \in \mathbb{F}[\mathbf{x}]$ and an integer d , we use $H_d(f)$ to denote the homogeneous degree- d component of f . We say that a polynomial f has *min-degree* t if $H_t(f) \neq 0$ and $H_i(f) = 0$ for all $i < t$, and we define the min-degree of the zero polynomial to be $+\infty$.

- **Theorem 4** (Adaptation of [4], see Section 5). *Let \mathbb{F} be an algebraically-closed field of characteristic $q > 0$. Let $f \in \mathbb{F}[x_1, \dots, x_n]$ be a non-zero symmetric polynomial such that the min-degree of f_{elem} is t . Furthermore, assume that $f_{\text{elem}}(y_1, \dots, y_n)$ does not depend on the variables y_{n-1} and y_n . Then $H_t(f_{\text{elem}}) \preceq_c f$.*

In the above statement we say that f_{elem} must not depend on the variables y_{n-1} and y_n . This is a mere technical condition required in our proof of this theorem. Finally, we also need the following standard fact:

- **Lemma 5** (Homogeneous component extraction. Folklore, see [17, 2]). *Let \mathbb{F} be any field. For any $f \in \mathbb{F}[\mathbf{x}]$ and integer $d \geq 0$, we have $H_d(f) \preceq_c f$.*

Permanents

The canonical VNP-complete polynomial family is given by the polynomials Per_n for $n \in \mathbb{N}$, each defined on n^2 variables $x_{i,j}$ for $i, j \in [n]$, such that

$$\text{Per}_n = \sum_{\sigma \in S_n} x_{1,\sigma(1)} \cdots x_{n,\sigma(n)},$$

where S_n is the set of all permutations of the set $\{1, 2, \dots, n\}$. When the variables $x_{i,j}$ take Boolean values, the underlying input to Per_n defines a bipartite graph and the above polynomial computes the number of perfect matchings in this graph.

An analogous polynomial can be defined for not necessarily bipartite graphs. Assume that n is an even integer and fix the set of $\binom{n}{2}$ variables $x_{\{i,j\}}$ for all distinct $i, j \in [n]$. Then, we define the *perfect matching polynomial* PerfMatch_n over these variables by

$$\text{PerfMatch}_n = \sum_{\substack{\text{perfect matchings} \\ M \text{ of } K_n}} \prod_{\{i,j\} \in M} x_{\{i,j\}}.$$

We can also define analogues of the above for *hypergraphs*. Let $k \geq 2$ be an integer and let $K_n^{(k)}$ denote the complete k -uniform hypergraph on n vertices. For n divisible by k , we define the *hypergraph perfect matching polynomial* $\text{hPerfMatch}_n^{(k)}$ over the $\binom{n}{k}$ many variables x_S for $S \in \binom{[n]}{k}$ by

$$\text{hPerfMatch}_n^{(k)} = \sum_{\substack{\text{perfect matchings} \\ M \text{ of } K_n^{(k)}}} \prod_{S \in M} x_S.$$

Note that $\text{PerfMatch}_n = \text{hPerfMatch}_n^{(2)}$.

We have the following simple reductions from permanents to their variants.

► **Lemma 6.** *For even $n \in \mathbb{N}$, we have $\text{Per}_{n/2} \preceq_c \text{PerfMatch}_n$. More generally, for any fixed $k \in \mathbb{N}$ and any n divisible by k , we have $\text{Per}_{n/k} \preceq_c \text{hPerfMatch}_n^{(k)}$.*

Proof sketch. For even n , reduce $\text{Per}_{n/2}$ to PerfMatch_n as follows: For $i, j \in [n/2]$, substitute $x_{\{i,n/2+j\}} \leftarrow x_{i,j}$ and $x_S \leftarrow 0$ for all remaining variables x_S . This results in $\text{Per}_{n/2}$.

More generally, for n divisible by k , reduce $\text{Per}_{n/k}$ to $\text{hPerfMatch}_n^{(k)}$ as follows: For $i, j \in [n/k]$, let $S_{i,j} = \{i\} \cup \{tn/k + j \mid t = 1, \dots, k - 1\}$ and substitute $x_{S_{i,j}} \leftarrow x_{i,j}$. Then substitute $x_S \leftarrow 0$ for all remaining variables x_S . This results in $\text{Per}_{n/k}$. ◀

Finally, we recall a generalization of the permanent to *rectangular matrices*. Fix an $r \times n$ matrix X where $r \leq n$ and the (i, j) -th entry of X is a variable $x_{i,j}$. For a subset $J \subseteq [n]$ of size r , we define X_J to be the submatrix obtained by keeping only the columns indexed by the indices in J . Now, we define the rectangular permanent $\text{rPer}_{r,n}$ by

$$\text{rPer}_{r,n} = \sum_{J \in \binom{[n]}{r}} \text{Per}_r(X_J).$$

The following polynomial identity will be crucial to our main results.

► **Theorem 7** (Binet-Minc Identity [12]). *Let \mathbb{F} be any field. Fix an $r \times n$ matrix X as above. For any non-empty $I \subseteq [n]$, define the polynomial S_I by $S_I = \sum_{j=1}^n \prod_{i \in I} x_{i,j}$. Then, we have*

$$\text{rPer}_{r,n} = \sum_{\mathcal{I} \in \mathcal{P}_r} (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I| - 1)! \cdot S_I,$$

where \mathcal{P}_r denotes the set of all partitions of $[r]$ into non-empty subsets.

Sidon sets and variants

Our hardness proofs for the monomial symmetric functions m_λ require certain conditions on λ : In Section 3, any unordered pair of numbers in λ must be uniquely identified from its sum, i.e., the parts in λ form a so-called *Sidon set*. Additionally, sums composed of the parts in λ are stratified by the number of terms involved in the sum. Section 4 requires more generally that sets of fixed size $q \in \mathbb{N}$ are identifiable, and that all parts must have remainder 1 modulo q . We capture these requirements in the following definition:

► **Definition 8.** *Given a set of integers $L = \{\lambda_1, \dots, \lambda_r\}$ and a subset $S \subseteq [r]$, define $\lambda_S := \sum_{i \in S} \lambda_i$. We say that L (or a partition λ whose multiset of parts equals L) is q -good for an integer $q \geq 2$ if the following conditions hold:*

q -wise Sidon set: For any two distinct sets $S, S' \subseteq [r]$ of size q , we have $\lambda_S \neq \lambda_{S'}$.

Stratification: For sets $S, T \subseteq [r]$ with $|S| < q$ and $|T| = q$, we have $\lambda_S < \lambda_T$.

Units modulo $q + 1$: For each $i \in [r]$, we have $\lambda_i \equiv 1 \pmod{q + 1}$.

Existing constructions of q -wise Sidon sets can be adapted to construct such sets:

► **Lemma 9.** *For all $r, q \in \mathbb{N}$, there exists a q -good set of r integers that are bounded by $r^{O(q)}$. Such a set can be constructed deterministically in time $r^{O(q)}$.*

Proof. Let $s \in \mathbb{N}$ be the smallest perfect square that is larger or equal to r . By Lemma 2.5 in [9], there is a q -wise Sidon set $\{\lambda_1, \dots, \lambda_s\}$ with elements bounded by $s^{O(q)} = r^{O(q)}$ that can be constructed in $s^{O(q)} = r^{O(q)}$ time. Then the r -element subset $\{\lambda_1, \dots, \lambda_r\}$ trivially is a q -wise Sidon set as well.

Now take $\mu_i = (q + 1)\lambda_i + 1$ for all $i \in [r]$; this trivially ensures that $\mu_i \equiv 1 \pmod{q + 1}$ for all i , as required in the third property from Definition 8. As the map $x \mapsto (q + 1)x + 1$ is injective, the set $\{\mu_1, \dots, \mu_r\}$ is a q -wise Sidon set.

Finally, to ensure the stratification property, let Σ be the smallest multiple of $q + 1$ that is strictly larger than $\mu_1 + \dots + \mu_r$, define $\mu'_i = \Sigma + \mu_i$ for $i \in [r]$, and set $L := \{\mu'_1, \dots, \mu'_r\}$. As the map $x \mapsto \Sigma + x$ is injective, L is a q -wise Sidon set. As Σ is a multiple of $q + 1$, we have $\mu'_i \equiv \mu_i \equiv 1 \pmod{q + 1}$ for all i . We show that $\mu'_I < \mu'_{I'}$ for $I, I' \subseteq [r]$ with $|I| < |I'|$: Note that μ'_i can be interpreted as a 2-digit number $(1, \mu_i)$ in base Σ . For $I \subseteq [r]$, the representation of $\mu'_I = \sum_{i \in I} \mu'_i$ in base Σ is $(|I|, \mu_I)$; this is because Σ is large enough to avoid an overflow of the least significant digit. The stratification property follows.

From the above construction, it follows that L is a q -good set, all numbers in L are bounded by $r^{O(q)}$, and that L can be constructed deterministically in $r^{O(q)}$ time. ◀

3 Main result in characteristic zero

We present our main reduction from permanents to monomial symmetric functions m_λ . The reduction shown in this section applies to the field \mathbb{C} . In the next section, we show how to handle fields of characteristic strictly greater than 2; this introduces additional technical difficulties that are not present in this section.

Fix a 2-good partition $\lambda = (\lambda_1, \dots, \lambda_r)$ with r parts, non-increasingly ordered, and $\lambda \vdash d$ for $d \in \mathbb{N}$. Recall our notation $\lambda_I := \sum_{i \in I} \lambda_i$ for $I \subseteq [r]$. We first express $m_\lambda(x_1, \dots, x_n)$ for $n \in \mathbb{N}$ as a polynomial combination of the power-sum symmetric polynomials $p_j := p_{n,j}(x_1, \dots, x_n)$ for $1 \leq j \leq d$. That is, we obtain a polynomial $f_{\text{pow}}(y_1, \dots, y_d)$ in indeterminates y_1, \dots, y_d such that

$$m_\lambda(x_1, \dots, x_n) = f_{\text{pow}}(p_1, \dots, p_d).$$

Known reductions will allow us to reduce directly (in characteristic 0) or with extra steps (for characteristic > 2) from f_{pow} to m_{λ} . It therefore remains to establish hardness of f_{pow} . Towards this, we give a combinatorial interpretation of f_{pow} as a sum over partitions of $[r]$; this sum will later be restricted to partitions that are actually perfect matchings of K_r .

► **Fact 10.** *If $\lambda = (\lambda_1, \dots, \lambda_r)$ is a partition of some integer $d \in \mathbb{N}$, and the parts of λ are pairwise distinct, then we have $m_{\lambda}(x_1, \dots, x_n) = f_{\text{pow}}(p_1, \dots, p_d)$ with*

$$f_{\text{pow}}(y_1, \dots, y_d) = \sum_{\mathcal{I} \in \mathcal{P}_r} (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I| - 1)! \cdot y_{\lambda_I}. \quad (1)$$

Proof. If all parts of λ are pairwise distinct, then m_{λ} can be expressed as the rectangular permanent of a generalized Vandermonde matrix V_{λ} defined from λ :

$$m_{\lambda} = \text{rPer}_{r,n} \underbrace{\begin{pmatrix} x_1^{\lambda_1} & x_2^{\lambda_1} & \dots & x_n^{\lambda_1} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{\lambda_r} & x_2^{\lambda_r} & \dots & x_n^{\lambda_r} \end{pmatrix}}_{=: V_{\lambda}} \quad (2)$$

The Binet-Minc formula (Theorem 7) then readily yields (1): When invoked on V_{λ} , the polynomial S_I in the statement of Theorem 7 equals

$$S_I = \sum_{j=1}^n \prod_{i \in I} V_{\lambda}(i, j) = \sum_{j=1}^n \prod_{i \in I} x_j^{\lambda_i} = \sum_{j=1}^n x_j^{\lambda_I} = p_{\lambda_I}.$$

This concludes the proof. ◀

Note that all parts of λ are indeed distinct, since λ is 2-good and thus cannot feature a part of multiplicity strictly larger than 1; this follows from the Sidon set property.

Theorem 2 shows that f_{pow} is uniquely determined over characteristic 0, and Theorem 3 yields a reduction from f_{pow} to m_{λ} , so we establish hardness of f_{pow} : We define a new polynomial f_{match} by restricting the sum over partitions $\mathcal{I} \in \mathcal{P}_r$ in (1) to perfect matchings, i.e., to partitions of $[r]$ in which all parts have cardinality 2. We write \mathcal{M}_r for the set of perfect matchings of $[r]$ and define

$$\begin{aligned} f_{\text{match}}(y_1, \dots, y_d) &:= \sum_{\mathcal{I} \in \mathcal{M}_r} (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I| - 1)! \cdot y_{\lambda_I} \\ &= (-1)^{r/2} \sum_{\mathcal{I} \in \mathcal{M}_r} \prod_{I \in \mathcal{I}} y_{\lambda_I}. \end{aligned} \quad (3)$$

The last identity holds because every $\mathcal{I} \in \mathcal{M}_r$ has exactly $r/2$ parts, each of cardinality 2.

We will show later that f_{match} can be reduced to f_{pow} . First, we establish the hardness of f_{match} by reducing the perfect matching polynomial to it. Here, we crucially use that λ is a Sidon set in order to switch between the variables $y_{\lambda_{\{u,v\}}}$ present in f_{match} and the variables $x_{\{u,v\}}$ present in PerfMatch_r .

▷ **Claim 11.** There is a c-reduction from PerfMatch_r to f_{match} .

Proof. Since λ is a 2-good set, its parts form a 2-wise Sidon set, so the map $\{u, v\} \mapsto \lambda_{\{u,v\}}$ from 2-subsets of $[r]$ into \mathbb{N} is injective. This in turn implies that substituting $y_{\lambda_{\{u,v\}}} \leftarrow x_{\{u,v\}}$ for all $\{u, v\} \subseteq [r]$ into f_{match} yields the polynomial

$$(-1)^{r/2} \sum_{\mathcal{I} \in \mathcal{M}_r} \prod_{I \in \mathcal{I}} x_{\{u,v\}} = (-1)^{r/2} \text{PerfMatch}_r.$$

Multiplication with $(-1)^{r/2}$ then yields the desired c-reduction. ◀

16:8 On the VNP-Hardness of Some Monomial Symmetric Polynomials

Finally, we reduce f_{match} to f_{pow} . This reduction proceeds in two steps: We first show that the homogeneous component of degree $r/2$ in f_{pow} enumerates the perfect matchings and some additional structures; these additional structures are then removed through the stratification property of λ .

▷ **Claim 12.** There is a c -reduction from f_{match} to f_{pow} .

Proof. Consider the homogeneous component $H_{r/2}(f_{\text{pow}})$ in f_{pow} . Lemma 5 gives a c -reduction from $H_{r/2}(f_{\text{pow}})$ to f_{pow} . By inspecting (1), we see that the monomials of $H_{r/2}(f_{\text{pow}})$ correspond to the partitions $\mathcal{I} \in \mathcal{P}_r$ with exactly $r/2$ parts. Such a partition is a perfect matching iff it contains no parts of size 1, as every part must then be of cardinality at least 2, and thus, of cardinality exactly 2.

We thus aim to restrict the sum further to partitions with $r/2$ parts and no parts of cardinality 1. To this end, substitute $p_{\lambda_{\{u\}}} \leftarrow 0$ for all $u \in [d]$: By the stratification property of λ , this eliminates precisely those partitions from $H_{r/2}(f_{\text{pow}})$ that contain a singleton part $\{u\}$. Overall, this yields a c -reduction from f_{match} over $H_{r/2}(f_{\text{pow}})$ to f_{pow} . ◁

We have now collected all parts of the reduction and summarize it below.

► **Lemma 13.** Let $\mathbb{F} = \mathbb{C}$. Let $\lambda \vdash d$ for $d \in \mathbb{N}$ be a 2-good partition with r parts. Then

$$\text{Per}_{r/2} \preceq_c m_\lambda(x_1, \dots, x_n)$$

provided that $n \geq d$.

Proof. Let $f_{\text{pow}}(y_1, \dots, y_d)$ and $f_{\text{match}}(y_1, \dots, y_d)$ denote the polynomials defined from λ in (1) and (3) above. We have the following chain of reductions:

$$\begin{aligned} \text{Per}_{r/2} &\preceq_c \text{PerfMatch}_r && \text{by Lemma 6} \\ &\preceq_c f_{\text{match}}(y_1, \dots, y_d) && \text{by Claim 12} \\ &\preceq_c f_{\text{pow}}(y_1, \dots, y_d) && \text{by Claim 11} \\ &\preceq_c m_\lambda(x_1, \dots, x_n) && \text{by Theorem 4.} \end{aligned}$$

The lemma follows. ◀

Combining Lemma 13 and Lemma 9, we obtain a proof of Theorem 1 in the case when the underlying field is \mathbb{C} .

Proof of Theorem 1 (characteristic 0). By Lemma 9, there is a sequence of 2-good partitions $\lambda_1, \lambda_2, \lambda_3, \dots$ such that $\lambda_n \vdash d_n$ has n parts and $d_n \leq s(n)$ for a polynomial $s : \mathbb{N} \rightarrow \mathbb{N}$. By Lemma 13, we have $\text{Per}_{n/2} \preceq_c m_{\lambda_n}(x_1, \dots, x_{s(n)})$. The theorem follows. ◀

4 Main result in positive characteristic

In this section, we adapt the proof from Section 3 to prove the main theorem for fields of positive characteristic. Throughout this section, \mathbb{F} denotes an infinite and algebraically closed field of characteristic $q > 2$. Rather than reducing from the perfect matching polynomial for graphs, we reduce from the perfect matching polynomial in $(q-1)$ -uniform hypergraphs. In the following, let λ be a $(q-1)$ -good partition with r parts and $\lambda \vdash d$ for $d \in \mathbb{N}$.

The proof begins again by expressing $m_{\lambda}(x_1, \dots, x_n) = f_{\text{pow}}(p_1, \dots, p_d)$ as a polynomial combination of power-sum polynomials p_i for $1 \leq j \leq d$. Since λ is $(q-1)$ -good, it contains only pairwise distinct parts, so we can use Fact 10 again and obtain

$$f_{\text{pow}}(y_1, \dots, y_d) = \sum_{\mathcal{I} \in \mathcal{P}_r} (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I|-1)! \cdot y_{\lambda_I}. \quad (4)$$

At this point, we exploit the field characteristic: We have $(|I|-1)! \equiv 0 \pmod{q}$ if $|I| > q$, implying that only partitions with parts of cardinality $\leq q$ appear in the above sum. Write $\mathcal{P}_r^{\leq q}$ for the set of these partitions, and furthermore write \mathcal{P}_r^{q-1} for the set of partitions whose parts all have cardinality $q-1$. Our goal is to restrict the sum in (4) to partitions from \mathcal{P}_r^{q-1} , that is, to perfect matchings in the complete $(q-1)$ -uniform r -vertex hypergraph. This resembles the restriction to graph perfect matchings in Section 3.

To achieve this restriction and to invoke Theorem 4 later, we express the power-sum polynomials p_k for $1 \leq k \leq d$ as polynomials in the elementary symmetric polynomials. In contrast to the converse direction (of expressing the elementary symmetric polynomials in terms of the power-sum polynomials), such expressions exist even in positive characteristic: For all $k \in \mathbb{N}$, there is a unique polynomial $f_k(z_1, \dots, z_k)$ with $p_k = f_k(e_1, \dots, e_k)$, even over fields of characteristic $q > 0$. Combined with (4), we obtain $m_{\lambda} = f_{\text{elem}}(e_1, \dots, e_d)$ with

$$f_{\text{elem}}(z_1, \dots, z_d) = \sum_{\mathcal{I} \in \mathcal{P}_r} (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I|-1)! \cdot f_{\lambda_I}(z_1, \dots, z_d). \quad (5)$$

The polynomial f_{elem} is unique, since the elementary symmetric polynomials form a basis for the symmetric polynomials over every field. Let t denote the min-degree of f_{elem} . Theorem 4 shows that the homogeneous component of degree t in f_{elem} admits a c -reduction to the polynomial m_{λ} , so we will focus on this homogeneous component. First, we show that the polynomial f_k , which expresses the power-sum symmetric polynomial p_k in terms of the elementary symmetric polynomials, has min-degree at least 2 whenever k is divisible by q . Note that f_k has no constant term.

▷ **Claim 14.** The only linear monomial in f_k is $(-1)^{k+1}k \cdot y_k$. In particular, if $q \mid k$, then the min-degree of f_k over characteristic q is at least 2.

Proof. Given a partition $\mu \vdash k$ and $i \in \mathbb{N}$, write $s_i(\mu)$ for the multiplicity of i in μ . We have [18, Chapter 7] that

$$f_k(y_1, \dots, y_k) = (-1)^k k \sum_{\mu \vdash k} \frac{(s_1(\mu) + s_2(\mu) + \dots + s_k(\mu) - 1)!}{s_1(\mu)! s_2(\mu)! \dots s_k(\mu)!} \prod_{i=1}^k (-y_i)^{s_i(\mu)}. \quad (6)$$

Note that every partition $\mu \vdash k$ with at least two parts contributes a term of total degree at least two. Only the partition $\mu = (k)$ can therefore contribute a linear monomial, and the contributed monomial is $(-1)^k k \cdot 0!/1! \cdot (-y_k) = (-1)^{k+1}k \cdot y_k$. ◁

Using this claim, we can analyze the min-degree of the contribution to f_{elem} from a partition $\mathcal{I} \in \mathcal{P}_r^{\leq q}$. That is, we write $f_{\text{elem}} = \sum_{\mathcal{I}} b_{\mathcal{I}}$ with \mathcal{I} ranging over $\mathcal{P}_r^{\leq q}$ and

$$b_{\mathcal{I}} := (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I|-1)! \cdot f_{\lambda_I}.$$

It turns out that the min-degree of $b_{\mathcal{I}}$ is minimized for partitions $\mathcal{I} \in \mathcal{P}_r^{q-1}$. This will allow us to isolate these partitions via Theorem 4.

16:10 On the VNP-Hardness of Some Monomial Symmetric Polynomials

▷ Claim 15. Let $\mathcal{I} \in \mathcal{P}_r^{\leq q}$.

- If $\mathcal{I} \in \mathcal{P}_r^{q-1}$, then the min-degree of $b_{\mathcal{I}}$ is equal to $r/(q-1)$.
- Otherwise, the min-degree of $b_{\mathcal{I}}$ is strictly larger than $r/(q-1)$.

Proof. Parts of size q in \mathcal{I} contribute 2 to the min-degree of $b_{\mathcal{I}}$, while parts of size $\leq q-1$ contribute 1. Consider a Knapsack instance \mathcal{K} with items S_1, \dots, S_q , and item repetitions allowed, where item S_j for $1 \leq j \leq q-1$ has weight 1 and profit j , while item S_q has weight 2 and profit q . The min-degree of $b_{\mathcal{I}}$ for $\mathcal{I} \in \mathcal{P}_r^{\leq q}$ can be viewed as the minimum weight of a solution with profit r for \mathcal{K} . Greedily choosing copies of the item S_{q-1} with strictly (since $q > 2$) largest profit-weight ratio yields an optimal fractional solution for \mathcal{K} that consists of $r/(q-1)$ copies of item S_{q-1} . This is an optimal *integral* solution to \mathcal{K} , and by optimality of the greedy algorithm, any solution including other items has strictly higher weight.

It follows that the min-degree of $b_{\mathcal{I}}$ over all $\mathcal{I} \in \mathcal{P}_r^{\leq q}$ is at least $r/(q-1)$, and this bound is attained with (and only with) the partitions $\mathcal{I} \in \mathcal{P}_r^{q-1}$. \triangleleft

It follows that the min-degree of f_{elem} is $t := r/(q-1)$. Since only partitions $\mathcal{I} \in \mathcal{P}_r^{q-1}$ have this min-degree t , the homogeneous component of degree t in f_{elem} depends only on these partitions. We obtain

$$H_t(f_{\text{elem}}) = H_t\left(\sum_{\mathcal{I} \in \mathcal{P}_r^{q-1}} b_{\mathcal{I}}\right) = H_t\left(\sum_{\mathcal{I} \in \mathcal{P}_r^{q-1}} (-1)^{r-|\mathcal{I}|} \prod_{I \in \mathcal{I}} (|I|-1)! \cdot f_{\lambda_I}\right). \quad (7)$$

Since all partitions $\mathcal{I} \in \mathcal{P}_r^{q-1}$ have t parts, each of size $q-1$, we obtain furthermore that

$$H_t(f_{\text{elem}}) = (-1)^{r-t}(q-2)! \cdot H_t\left(\sum_{\mathcal{I} \in \mathcal{P}_r^{q-1}} \prod_{I \in \mathcal{I}} f_{\lambda_I}\right). \quad (8)$$

The min-degree of f_{λ_I} for $I \in \mathcal{I} \in \mathcal{P}_r^{q-1}$ is 1, and the unique linear monomial is $(-1)^{\lambda_I+1} \lambda_I \cdot y_{\lambda_I}$. Since λ is $(q-1)$ -good and $|I| = q-1$, we have $\lambda_I \equiv q-1 \pmod{q}$. It follows that

$$H_1(f_{\lambda_I}) \equiv (-1)^q (q-1) \cdot y_{\lambda_I} \pmod{q} \quad (9)$$

For $I \in \mathcal{P}_r^{q-1}$, the degree- t homogeneous component of $\prod_{I \in \mathcal{I}} f_{\lambda_I}$ is the product of these linear monomials $H_1(f_{\lambda_I})$. That is,

$$H_t\left(\prod_{I \in \mathcal{I}} f_{\lambda_I}\right) \equiv \prod_{I \in \mathcal{I}} H_1(f_{\lambda_I}) \equiv (-1)^{(q+1)t} \prod_{I \in \mathcal{I}} y_{\lambda_I} \pmod{q} \quad (10)$$

It follows that

$$H_t(f_{\text{elem}}) \equiv (-1)^{r-t+(q+1)t} (q-2)! \sum_{\mathcal{I} \in \mathcal{P}_r^{q-1}} \prod_{I \in \mathcal{I}} y_{\lambda_I} \pmod{q} \quad (11)$$

Using the $(q-1)$ -wise Sidon set property of λ , we can substitute $y_{\lambda_I} \leftarrow x_I$ for all sets $I \subseteq [r]$ of cardinality $q-1$ into (11) as in Claim 11, so as to obtain:

▷ Claim 16. The polynomial $\text{hPerfMatch}_r^{q-1}$ admits a c -reduction to $H_t(f_{\text{elem}})$.

It remains to invoke Theorem 4. We collect the proof steps in the following lemma that parallels Lemma 13 for characteristic 0.

► **Lemma 17.** *Let \mathbb{F} be an algebraically closed field of characteristic $q > 2$. Let $\lambda \vdash d$ for $d \in \mathbb{N}$ be a $(q-1)$ -good partition with r parts. Then*

$$\text{Per}_{r/(q-1)} \preceq_c m_\lambda(x_1, \dots, x_n),$$

provided that $n \geq d + 2$.

Proof. Let $f_{\text{elem}}(y_1, \dots, y_d)$ denote the polynomial defined from λ in (5). We have the following chain of reductions:

$$\begin{aligned} \text{Per}_{r/(q-1)} &\preceq_c \text{hPerfMatch}_r^{(q-1)} && \text{by Lemma 6} \\ &\preceq_c H_t(f_{\text{elem}}(y_1, \dots, y_d)) && \text{by Claim 16} \\ &\preceq_c m_\lambda(x_1, \dots, x_n) && \text{by Theorem 4.} \end{aligned}$$

To invoke Theorem 4, we use that $n \geq d + 2$. This means that indeed $f_{\text{elem}}(y_1, \dots, y_d)$ depends on two variables less than $m_\lambda(x_1, \dots, x_n)$, as required. ◀

The proof of Theorem 1 for characteristic q now follows as in Section 3: Use Lemma 9 to find $(q-1)$ -good partitions, then reduce from the family of permanents via Lemma 17.

5 Proof of Theorem 4

In this section, we outline how to modify the result of [4] to show Theorem 4 over an algebraically closed field \mathbb{F} of any characteristic (we will only require that the size of the field \mathbb{F} is large enough and contains primitive roots of unity of large enough order).

High-level Idea

The modification is based on a very simple idea. [4] prove a result for any algebraically independent polynomials satisfying a (simple) technical condition. To apply this result, the underlying field is required to have characteristic zero in order to apply the *Jacobian criterion*, which states that the Jacobian of a collection of algebraically independent polynomials is full rank over fields of characteristic zero. While this fact fails for fields of positive characteristic, the proof still works if we are independently able to show that the polynomials under consideration induce a Jacobian of full rank. We use this fact to prove their result in the setting that the underlying polynomials are the elementary symmetric polynomials e_1, \dots, e_{n-2} .

The following is implicit in [4, Lemma 27]. The proof is only stated for homogeneous polynomials g but easily works in the following more general setting as well.

► **Lemma 18.** *Let k, n be positive integers with $k \leq n$. Assume that $Q_1, \dots, Q_k \in \mathbb{F}[x_1, \dots, x_n]$ are polynomials of degree at most D such that for some $\mathbf{a} \in \mathbb{F}^n$, we have*

- $Q_1(\mathbf{a}) = \dots = Q_k(\mathbf{a}) = 0$, and
- the $k \times n$ Jacobian matrix $\mathcal{J}(Q_1, \dots, Q_k)$ has rank k , when evaluated at the point \mathbf{a} .

Further, assume that $g \in \mathbb{F}[y_1, \dots, y_k]$ is a degree- d polynomial of min-degree t and let $G = g(Q_1, \dots, Q_k)$. Then, $L^G(H_t(g)) \leq \text{poly}(n, d, D)$.

We only sketch the proof, as it is quite similar to [4, Lemma 27].

Proof sketch. By shifting the input \mathbf{x} by \mathbf{a} , we assume without loss of generality that \mathbf{a} is the origin (note that this does not affect the Jacobian at all). Now, by a Taylor expansion around the origin, we have for each $i \in [k]$

$$Q_i(\mathbf{x}) = \ell_i(\mathbf{x}) + R_i(\mathbf{x})$$

16:12 On the VNP-Hardness of Some Monomial Symmetric Polynomials

where $\ell_i(\mathbf{x})$ is a homogeneous linear polynomial and $R_i(\mathbf{x})$ is a polynomial of min-degree at least 2. Further, the polynomials ℓ_1, \dots, ℓ_k are linearly independent as the Jacobian is full-rank at \mathbf{a} (i.e. the origin). Thus, we have

$$\begin{aligned} G(\mathbf{x}) &= g(Q_1(\mathbf{x}), \dots, Q_k(\mathbf{x})) \\ &= \sum_{j=t}^d H_j(g)(\ell_1(\mathbf{x}) + R_1(\mathbf{x}), \dots, \ell_k(\mathbf{x}) + R_k(\mathbf{x})) \\ &= H_t(g)(\ell_1(\mathbf{x}), \dots, \ell_k(\mathbf{x})) + R(\mathbf{x}) \end{aligned}$$

where $R(\mathbf{x})$ has min-degree strictly greater than t and degree at most $\deg(G)$. Note that the second equality uses the fact that the min-degree of g is t . Since ℓ_1, \dots, ℓ_k are linearly independent, there exists a homogeneous linear transformation T of the variables x_1, \dots, x_n such that $\ell_i(T(\mathbf{x})) = x_i$ for each $i \in [k]$. Applying this linear transformation to the input variables, we have

$$G'(\mathbf{x}) := G(T(\mathbf{x})) = H_t(g)(\ell_1(T(\mathbf{x})), \dots, \ell_k(T(\mathbf{x}))) + R(T(\mathbf{x})) = H_t(g)(x_1, \dots, x_k) + R'(\mathbf{x})$$

where R' has min-degree strictly greater than t and degree at most $\deg(G)$.

The above clearly implies that $L^G(G') \leq \text{poly}(n)$. Furthermore, by Lemma 5, we have that $L^{G'}(H_t(g)) \leq \text{poly}(n, \deg(G)) \leq \text{poly}(n, d, D)$ as the degree of G is at most $d \cdot D$.

Composing the two reductions, we have $L^G(H_t(g)) \leq \text{poly}(n, d, D)$. \blacktriangleleft

We will apply Lemma 18 to the setting when Q_1, \dots, Q_k are e_1, \dots, e_k for some $k < n - 1$. To do this, we need to show that these polynomials satisfy the hypotheses required of Q_1, \dots, Q_k in the statement of Lemma 18. We do this now, using ideas from Lemma 30 and 31 of [4].

► Lemma 19. *Let k, n be positive integers with $k < n - 1$. Then the polynomials e_1, \dots, e_k satisfy the conditions required of Q_1, \dots, Q_k in the hypothesis of Lemma 18.*

Proof sketch. Define $\ell = k + 1$ if q does not divide $k + 1$ and $\ell = k + 2$ otherwise. Note that $k < \ell \leq n$. As q does not divide ℓ , the algebraically-closed field \mathbb{F} contains ℓ distinct ℓ -th roots of unity $1, \omega, \dots, \omega^{\ell-1}$. Let $\mathbf{a} = (1, \omega, \dots, \omega^{\ell-1}, 0, \dots, 0)$. It is a standard observation (see e.g. [4, Lemma 31]) that $e_1(\mathbf{a}) = \dots = e_{\ell-1}(\mathbf{a}) = 0$. As $\ell > k$, this implies the first hypothesis from the statement of Lemma 18 above.

For the second hypothesis, we consider the Jacobian matrix $\mathcal{J}(e_1, \dots, e_k)$. To show that this matrix is full-rank when evaluated at \mathbf{a} , it suffices to argue that some $k \times k$ minor of this matrix is non-zero when evaluated at \mathbf{a} . We consider the minor J_k defined by the first k columns of $\mathcal{J}(e_1, \dots, e_k)$ (containing the partial derivatives w.r.t. variables x_1, \dots, x_k).

The proof of Lemma 30 in [4] shows that J_k is divisible by the polynomial $\prod_{i < j \leq k} (x_i - x_j)$. By comparing the degrees of these polynomials, we see immediately that J must be $c \cdot \prod_{i < j \leq k} (x_i - x_j)$ for some scalar $c \in \mathbb{F}$. As the first k co-ordinates of \mathbf{a} are distinct, we see that $J_k(\mathbf{a}) = c \cdot \alpha$ for some non-zero $\alpha \in \mathbb{F}$. So it suffices to show that c is non-zero.

To argue this, we only need to show that J_k is a non-zero polynomial. To see this, consider the coefficient of $x_1^{k-1} x_2^{k-2} \dots x_{k-1}$ in the minor J_k . We claim that this coefficient is non-zero. In particular, this implies that J_k is a non-zero polynomial.

It remains to prove the claim regarding the monomial $\mathbf{m}_k := x_1^{k-1} x_2^{k-2} \dots x_{k-1}$. We have

$$J_k = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^k \mathcal{J}(e_1, \dots, e_k)_{i, \sigma(i)}.$$

To argue that \mathbf{m}_k has a non-zero coefficient in J_k , we can argue by induction on k . Note that the (i, j) th entry of $\mathcal{J}(e_1, \dots, e_k)$ is the partial derivative of the polynomial e_i w.r.t. variable x_j . It is thus the sum of all multilinear monomials of degree $i - 1$ not divisible by x_j . In particular, the only entry in the k th row that has a monomial involving only the variables x_1, \dots, x_{k-1} (the set of variables of \mathbf{m}_k) is the entry $\mathcal{J}(e_1, \dots, e_k)_{k,k}$, and furthermore, the unique such monomial is $x_1 \cdots x_{k-1}$.

Expanding the determinant J_k by the Laplace expansion along the k th row, we see that the coefficient of \mathbf{m}_k in J_k is also the coefficient of \mathbf{m}_k in

$$x_1 \cdots x_{k-1} \cdot J'_k$$

where the latter term J'_k represents the co-factor of $\mathcal{J}(e_1, \dots, e_k)_{k,k}$ in J_k , which is exactly the minor corresponding to the first $k - 1$ columns of $\mathcal{J}(e_1, \dots, e_{k-1})$, which is J_{k-1} . By induction, the coefficient of $\mathbf{m}_{k-1} = x_1^{k-2} \cdots x_{k-2}$ in J'_k is non-zero, hence implying that the coefficient of \mathbf{m}_k in J_k is non-zero as well. ◀

To prove Theorem 4, we apply Lemma 18 to the case when $G = f(x_1, \dots, x_n)$ and $g = f_{\text{elem}}(y_1, \dots, y_{n-2})$. Note that, by the hypothesis of Theorem 4, f_{elem} does not depend on y_{n-1} and y_n . By Lemma 19, the polynomials e_1, \dots, e_{n-2} satisfy the hypotheses of Lemma 18. Applying the latter lemma and using the fact that e_1, \dots, e_{n-2} have degree at most n , we immediately get $H_t(f_{\text{elem}}) \preceq_c f$, implying Theorem 4.

References

- 1 Jayadev Acharya, Hirakendu Das, Alon Orlitsky, and Ananda Theertha Suresh. A unified maximum likelihood approach for estimating symmetric properties of discrete distributions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 11–21. PMLR, 2017. URL: <http://proceedings.mlr.press/v70/acharya17a.html>.
- 2 Markus Bläser and Gorav Jindal. On the complexity of symmetric polynomials. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPICs*, pages 47:1–47:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ITCS.2019.47.
- 3 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997.
- 4 Prasad Chaugule, Mrinal Kumar, Nutan Limaye, Chandra Kanta Mohapatra, Adrian She, and Srikanth Srinivasan. Schur polynomials do not have small formulas if the determinant doesn't. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPICs*, pages 14:1–14:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.14.
- 5 Sergey Fomin, Dima Grigoriev, and Gleb A. Koshevoy. Subtraction-free complexity, cluster transformations, and spanning trees. *Found. Comput. Math.*, 16(1):1–31, 2016. doi:10.1007/s10208-014-9231-y.
- 6 Hervé Fournier, Nutan Limaye, Meena Mahajan, and Srikanth Srinivasan. The shifted partial derivative complexity of elementary symmetric polynomials. *Theory Comput.*, 13(1):1–34, 2017. doi:10.4086/toc.2017.v013a009.
- 7 Dima Grigoriev and Gleb A. Koshevoy. Complexity of tropical schur polynomials. *J. Symb. Comput.*, 74:46–54, 2016. doi:10.1016/j.jsc.2015.05.005.
- 8 Pavel Hrubes and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Comput. Complex.*, 20(3):559–578, 2011. doi:10.1007/s00037-011-0007-3.

- 9 Mrinal Kumar and Ben Lee Volk. Lower bounds for matrix factorization. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPICs*, pages 5:1–5:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.5.
- 10 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 804–814. IEEE, 2021. doi:10.1109/FOCS52979.2021.00083.
- 11 I. G. (Ian Grant) Macdonald. *Symmetric functions and Hall polynomials*. Oxford mathematical monographs. Clarendon Press ; Oxford University Press, Oxford : New York, 1979.
- 12 Henryk Minc and Marvin Marcus. *Permanents*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1984. doi:10.1017/CB09781107340688.
- 13 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complexity*, 6(3):217–234, 1996/97. doi:10.1007/BF01294256.
- 14 Amritanshu Prasad. *Representation theory*, volume 147 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Delhi, 2015. A combinatorial viewpoint. doi:10.1017/CB09781139976824.
- 15 Amir Shpilka. Affine projections of symmetric polynomials. *J. Comput. Syst. Sci.*, 65(4):639–659, 2002. doi:10.1016/S0022-0000(02)00021-1.
- 16 Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Comput. Complex.*, 10(1):1–27, 2001. doi:10.1007/PL00001609.
- 17 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.
- 18 Richard P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999. With a foreword by Gian-Carlo Rota and appendix 1 by Sergey Fomin. doi:10.1017/CB09780511609589.

Online Piercing of Geometric Objects

Minati De ✉ 

Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Saksham Jain ✉

Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Sarat Varma Kallepalli ✉

Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Satyam Singh ✉

Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Abstract

We consider the online version of the piercing set problem where geometric objects arrive one by one. The online algorithm must maintain a piercing set for the arrived objects by making irrevocable decisions. First, we show that any deterministic online algorithm that solves this problem has a competitive ratio of at least $\Omega(n)$, which even holds when the objects are one-dimensional intervals. On the other hand, piercing unit objects is equivalent to the unit covering problem which is well-studied in the online model. Due to this, all the results related to the online unit covering problem are preserved for the online unit piercing problem when the objects are translated from each other. Surprisingly, no upper bound was known for the unit covering problem when unit objects are anything other than balls and hypercubes. In this paper, we introduce the notion of α -aspect and α -aspect $_{\infty}$ objects. We give an upper bound of competitive ratio for α -aspect and α -aspect $_{\infty}$ objects in \mathbb{R}^3 and \mathbb{R}^d , respectively, with a scaling factor in the range $[1, k]$. We also propose a lower bound of the competitive ratio for bounded scaled objects like α -aspect objects in \mathbb{R}^2 , axis-aligned hypercubes in \mathbb{R}^d , and balls in \mathbb{R}^2 and \mathbb{R}^3 . For piercing α -aspect $_{\infty}$ objects in \mathbb{R}^d , we show that a simple deterministic algorithm achieves a competitive ratio of at most $\left(\frac{2}{\alpha}\right)^d \left((1 + \alpha)^d - 1\right) \left(\lceil \log_{(1+\alpha)}\left(\frac{2k}{\alpha}\right) \rceil\right) + 1$. This result is very general in nature. One can obtain upper bounds for specific objects by specifying the value of α . By putting the value of $k = 1$ to the above result, we get an upper bound of the competitive ratio for the unit covering problem for various types of objects.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases piercing set problem, online algorithm, competitive ratio, unit covering problem, geometric objects

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.17

Related Version *Full Version:* <https://web.iitd.ac.in/~minati/archive/OnlinePiercing.pdf>

Funding Work on this paper by M. De has been partially supported by SERB MATRICS Grant MTR/2021/000584, and work by S. Singh has been supported by CSIR (File Number-09/086(1429)/2019-EMR_I).

Acknowledgements Authors would like to acknowledge anonymous reviewers for giving constructive feedback that helped to improve the presentation of the paper.

1 Introduction

Piercing is one of the most important problems in computational geometry. A set $\mathcal{P} \subset \mathbb{R}^d$ of points is defined as a *piercing set* for a set \mathcal{S} of geometric objects in \mathbb{R}^d , if each object in \mathcal{S} contains at least one point from \mathcal{P} . Given the set \mathcal{S} of objects, the piercing problem is to find a piercing set $\mathcal{P} \subset \mathbb{R}^d$ of minimum cardinality.



© Minati De, Saksham Jain, Sarat Varma Kallepalli, and Satyam Singh;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 17; pp. 17:1–17:16



Leibniz International Proceedings in Informatics

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

In this paper, we study the *online version* of the problem in which geometric objects arrive one by one, and the online algorithm needs to maintain a piercing set for the already arrived objects. Once a new geometric object σ arrives, the online algorithm needs to place a point $p \in \sigma$ if the existing piercing set does not already pierce it. An online algorithm may add points to the piercing set, but it cannot remove points from it, i.e., the online algorithm needs to take irrevocable decisions. The goal is to minimize the cardinality of the piercing set. We analyze the quality of our online algorithm by competitive analysis [3]. The *competitive ratio* of an online algorithm is $\sup_{\beta} \frac{ALG_{\beta}}{OPT_{\beta}}$, where β is an input sequence, and OPT_{β} and ALG_{β} are the cost of an optimal offline algorithm and the solution produced by the online algorithm, respectively, for the input sequence β . Here, the supremum is taken over all possible input sequences β .

The online version of this problem is inspired by an application in wireless networks [12]. Here the points model base stations, and the centers of objects model clients. The reception range of each client has some geometric shape (e.g., disks, hexagons, etc.). The algorithm needs to place a base station that serves a new uncovered client. Since installing the base station is expensive, so the overall goal is to place the minimum number of base stations.

1.1 Related Works

In the offline setting, the piercing set problem is well-studied problem [6, 11, 18]. For one-dimensional intervals, the minimum piercing set can be found in polynomial time, i.e., $O(n \log c^*)$, where c^* is the size of the minimum piercing set [19]. On the other hand, computing the minimum piercing set is NP-complete even for unit squares in the plane [15, 13]. Chan [6] proposed a polynomial-time approximation scheme for α -fat convex objects. Katz et al. [18] studied the problem in the dynamic setting for intervals.

A closely related well explored problem is the set-cover problem. Let X be a set of n elements, and let \mathcal{S} be a family of subsets of X such that $|\mathcal{S}| = m$. A *set cover* is a collection of subsets $\mathcal{S}^* \subset \mathcal{S}$ such that their union covers the whole set X . The goal of the *set cover problem* is to find a set cover of minimum cardinality. Note that by interchanging the roles of subsets and points, the set-cover problem and the hitting-set problems are equivalent. Both problems are classical NP-hard problems [17]. In the offline setting, the minimum hitting set problem for intervals (points in \mathbb{R}) can be solved in polynomial time using a greedy algorithm. However, these problems remain NP-hard, even for unit disks or unit squares in \mathbb{R}^2 [13]. Alon et al. [1] initiated the study of the online set-cover problem. In their setting, the sets X and \mathcal{S} are already known, but the order of arrivals of points is unknown. Upon the arrival of an uncovered point, the online algorithm must choose a subset that covers that point. The online algorithm presented by Alon et al. [1] achieves a competitive ratio of $O(\log n \log m)$. The results obtained in [1] also hold for the online hitting set problem. Even and Smorodinsky [12] studied the hitting set problem in an online setting, where both \mathcal{S} and X are known in advance, but the order of arrivals of the input objects in \mathcal{S} is unknown. In this setting, they proposed an algorithm with a competitive ratio of $O(\log n)$ for half-planes or unit disks. They also gave a matching lower bound of the competitive ratio for these cases. In the same paper, they proposed an online algorithm that achieves a tight bound of $\Theta(\log n)$ when objects are intervals and points are integers. Note that in our paper, the set $X = \mathbb{R}^d$, and the set \mathcal{S} is not known in advance.

A related problem is the *unit covering* (a special variant of the set cover problem), where the objective is to cover a given set of n points in \mathbb{R}^d with the minimum number of translated copies of an object. Charikar et al. [8] studied the problem in the online setting for unit balls in \mathbb{R}^d , where the points arrive one by one to the online algorithm, and the objective

is to assign a newly arrived point to an existing unit ball or a new unit ball to cover it. They proposed a deterministic online algorithm with a competitive ratio of $O(2^d d \log d)$ and show that the lower bound of the competitive ratio for this problem is $\Omega(\log d / \log \log \log d)$. Dumitrescu et al. [9] improved both the upper and lower bound of the problem to $O(1.321^d)$ and $\Omega(d + 1)$, respectively. They also proposed a lower bound of the competitive ratio for any centrally symmetric convex object as the illumination number (defined in Section 2) of the object. Recently, Dumitrescu and Tóth [10] proved that the competitive ratio of any deterministic online algorithm for the unit covering problem under L_∞ norm is at least 2^d . Surprisingly, we did not find any upper bound of the competitive ratio when objects are anything other than balls and hypercubes. In this paper, we raise this question and obtain a very general upper bound for the same.

Another relevant problem is the *chasing convex objects*, initiated by Friedman and Linal [14]. Here, convex objects $\sigma_i \subseteq \mathbb{R}^d$ arrives to the online algorithm, and the algorithm needs to place a point $p_i \in \sigma_i$ such that the total distances between successive points $\sum_{i=1}^{n-1} \|p_i - p_{i+1}\|$ is minimized, where n is length of input sequence. They proved that no online algorithm could achieve a competitive ratio lower than \sqrt{d} , and gave an online algorithm for $d = 2$. Bubeck et al. [5] presented an online algorithm with a competitive ratio $2^{O(d)}$ for this problem. Parallel to this, Bubeck et al. [4] presented an online algorithm having competitive ratio $O(\min(d, \sqrt{d \log n}))$ using the notion of Steiner point when objects are nested, i.e., $\sigma_1 \supseteq \sigma_2 \supseteq \dots \supseteq \sigma_n$. Later, Sellke [21] and Argue et al. [2] independently improved the result by showing that there exists an online algorithm having a competitive ratio of $O(\sqrt{d \log n})$ and $O(\min(d, \sqrt{d \log n}))$, respectively, for the chasing convex object problem.

1.2 Our Contributions

First, we prove that the competitive ratio of every deterministic online algorithm for piercing objects is at least $\Omega(n)$, which holds even when the input objects are one-dimensional intervals (Theorem 1, Section 2.1). As a result, next, we concentrate on when input objects are translated copies of an object. We show that for the translates of a centrally symmetric convex object C , the competitive ratio of any deterministic online algorithm is at least the illumination number of the object C (Theorem 2, Section 2.1). Next, we show that piercing translated copies of an object is equivalent to the unit covering problem (Theorem 6, Section 2.2). As an implication, all the results for the online unit covering problem obtained in [9] and [10] would be applicable to the unit piercing problem. Motivated by these, we ask what would happen when objects are neither of arbitrary size nor of unit size, but something in between: bounded scaled, and the shape of the object could vary. For this, we introduce the concept of α -aspect and α -aspect $_\infty$ objects in \mathbb{R}^d (Section 2.3). The α -aspect objects are invariant under translation, rotation and scaling. Whereas, α -aspect $_\infty$ objects are invariant under translation and scaling. For any object in \mathbb{R}^d , the value of α is in the interval $(0, 1]$. We consider objects with scaling factors in the range $[1, k]$ for any fixed $k \in \mathbb{R}$.

1. First, we prove that for piercing α -aspect $_\infty$ objects in \mathbb{R}^d , there exists a deterministic online algorithm whose competitive ratio is at most $\left(\frac{2}{\alpha}\right)^d \left((1 + \alpha)^d - 1\right) \left(\lceil \log_{(1+\alpha)}\left(\frac{2k}{\alpha}\right) \rceil\right) + 1$ (Theorem 7, Section 3).
2. Next, we show that for piercing α -aspect objects in \mathbb{R}^3 , there exists an online algorithm that achieves a competitive ratio of at most $\frac{2}{1 - \cos(\theta)} \lceil \log_{(1+x)}\left(\frac{2k}{\alpha}\right) \rceil + 1$, where $\theta = \frac{1}{2} \cos^{-1}\left(\frac{1}{2} + \frac{1}{1 + \sqrt{1 + 4\alpha^2}}\right)$ and $x = \frac{\sqrt{1 + 4\alpha^2} - 1}{2}$ (Theorem 12, Section 3). We achieve a similar result for piercing α -aspect objects in \mathbb{R}^2 (Theorem 16, Section 3).

3. Then, we prove that the competitive ratio of every deterministic online algorithm for piercing α -aspect objects in \mathbb{R}^2 is at least $\lfloor \log_{2/\alpha}(k) \rfloor + 1$ (Theorem 19, Section 4).
4. Later, we consider the case of axis-aligned d -dimensional hypercube. We show the competitive ratio of every deterministic online algorithm for piercing axis-aligned hypercubes in \mathbb{R}^d is at least $d(\lfloor \log_2(k) \rfloor) + 2^d$ (Theorem 21, Section 4).
5. At last, we consider balls in \mathbb{R}^3 (resp., \mathbb{R}^2), and we propose that the competitive ratio of every deterministic online algorithm for piercing balls in \mathbb{R}^3 (respectively, \mathbb{R}^2) with radius in the range $[1, k]$ is at least $3\lfloor \log_{(4+\epsilon)}(k) \rfloor + 4$ (respectively $2\lfloor \log_{(\frac{8}{3}+\epsilon)}(k) \rfloor + 3$), where ϵ is a very small constant close to zero (Theorem 22, Section 4).

To obtain the upper bound, we consider a very natural algorithm, ALGORITHM-CENTER, that works as follows. On receiving a new input object σ , if it is not pierced by the existing piercing set, our online algorithm adds the center of σ to the piercing set. For a hypercube σ , *center* is a point in σ from which maximum distance from any point of σ is minimized. For α -aspect and α -aspect $_\infty$ objects, a generalized definition of the center is given in Section 2.3.

1.3 Organization

In Section 2, initially, we propose a lower bound for piercing arbitrarily sized intervals followed by a lower bound for piercing a set of translated copies of a centrally symmetric convex object. Then, we discuss the equivalence of online unit piercing and online unit covering problems for translated copies of the unit object. At last, in this section, we introduce α -aspect objects and α -aspect $_\infty$ objects. In Section 3, we present an upper bound for α -aspect and α -aspect $_\infty$ objects having width in the range $[1, k]$. Later, in Section 4, we propose lower bounds for various bounded scaled objects like two-dimensional α -aspect objects, d -dimensional axis-aligned hypercubes in \mathbb{R}^d , and two and three-dimensional balls. Finally, in Section 5, we conclude.

2 Notation and Preliminaries

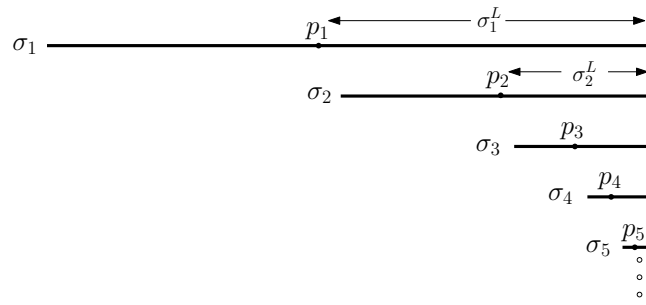
We use $[n]$ to denote the set $\{1, 2, \dots, n\}$. If not explicitly mentioned, we use the term *object* to denote a simply connected compact set in \mathbb{R}^d with a nonempty interior. The interior of an object C is represented by $\text{int}(C)$. A convex object is said to be a *centrally symmetric* if it is invariant under point reflection through its center. The *illumination number* of an object C , denoted by $I(C)$, is the minimum number of smaller homothetic copies of C whose union contains C [20].

2.1 Lower Bounds

First, we observe that the competitive ratio of the piercing problem has a pessimistic lower bound of $\Omega(n)$, which even holds for one-dimensional intervals.

► **Theorem 1.** *The competitive ratio of every deterministic online algorithm for piercing intervals is at least $\Omega(n)$, where n is the length of the input sequence.*

Proof. Let σ_1 be the first interval presented by the adversary to the online algorithm. Let p_1 be a point placed by the online algorithm to pierce the interval σ_1 . The point p_1 partitions the interval σ_1 into two parts, of which, let σ_1^L be a larger part that does not contain the point p_1 . Now, the adversary can place an interval σ_2 completely contained in σ_1^L (see Figure 1). For the new interval σ_2 , any online algorithm needs a new piercing point p_2 . Now



■ **Figure 1** Input instance of intervals.

again, one can define a partition σ_2^L of σ_2 depending on the position of the point p_2 such that σ_2^L does not contain p_2 and the adversary will place an interval σ_3 completely contained in σ_2^L . In this way, the adversary can adaptively construct n intervals for which any online algorithm needs n distinct points to pierce, while the offline optimum needs only one point. Hence, the lower bound of the competitive ratio is $\Omega(n)$. ◀

Now, we consider when objects are translated copies of a centrally symmetric convex object.

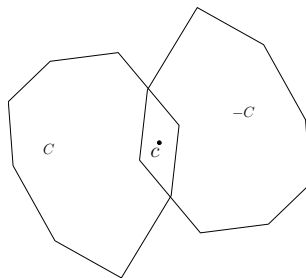
► **Theorem 2.** *The competitive ratio of every deterministic online algorithm for piercing a set of translated copies of a centrally symmetric convex object C is at least $I(C)$, where $I(C)$ denotes the illumination number of C .*

The proof can be adapted from [9, Thm 4]. Also, note that Theorem 2 can be obtained as a result of Theorem 6 and [9, Thm 4]. It is known that $I(C) = 2^d$ for any full-dimensional parallelepiped in \mathbb{R}^d [9, 20]. Consequently, the value of $I(C) = 2^d$, for d -dimensional hypercube. Therefore, we have the following.

► **Corollary 3.** *The competitive ratio of every deterministic online algorithm for piercing a set of axis-parallel unit hypercubes in \mathbb{R}^d is at least 2^d .*

2.2 Unit Covering vs Unit Piercing

Let C be an object, and let $-C$ be the reflection of C through a point $c \in C$ (see Figure 2).



■ **Figure 2** Reflection of an object C through a point c .

► **Definition 4 (Unit Piercing Problem).** *Given a family \mathcal{S} of translated copies of an object C , in the unit piercing problem, we need to pierce each object by placing the minimum number of points in \mathbb{R}^d .*

17:6 Online Piercing of Geometric Objects

► **Definition 5** (Unit Covering Problem). *Given a set of points $\mathcal{P} \subseteq \mathbb{R}^d$, in the unit covering problem, we need to place the minimum number of translated copies of an object $-C$ to cover all the points in \mathcal{P} .*

The following theorem connects the above two problems.

► **Theorem 6.** *The unit piercing problem is equivalent to the unit covering problem.*

As a consequence of Theorem 6, all the results related to the online unit covering problem (summarized in Table 1) studied in [9] and [10] are carried for the online piercing problem when objects are translates of each other.

■ **Table 1** Summary of known results for unit piercing problem.

Geometric Objects	Lower Bound	Upper Bound
Unit Intervals	2 [8]	2 [7]
Fixed oriented unit squares	4 [9, 10]	4 [7]
Fixed oriented unit hypercubes in \mathbb{R}^d	2^d [9, 10]	2^d [7, 10]
Unit disks in \mathbb{R}^2	4 [9]	5 [9]
Unit balls in \mathbb{R}^3	5 [9]	12 [9]
Unit balls in \mathbb{R}^d , $d > 3$	$d + 1$ [9]	$O(1.321^d)$ [9]
Translated copies of a centrally symmetric convex object C	$I(C)$ [9]	★ [Corollary 11]

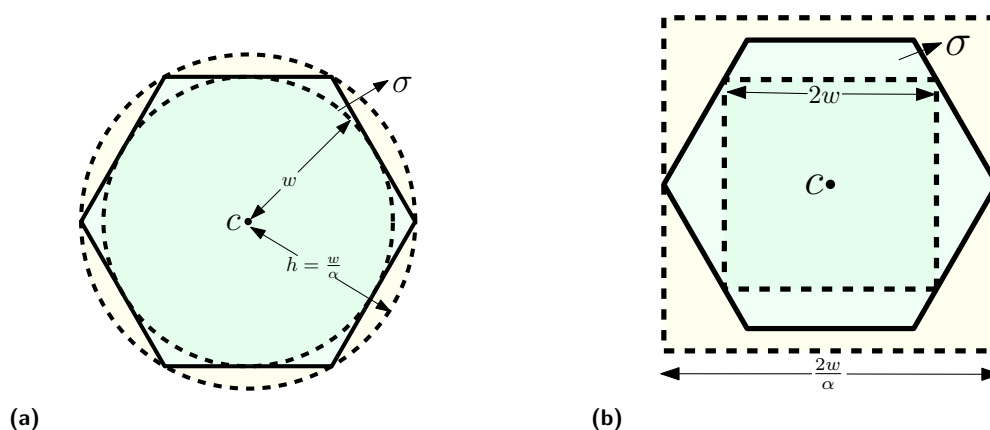
★ Result obtained in this paper.

2.3 Objects with Fixed Aspect Ratio

Let σ be an object and x be any point in σ . Let $\alpha(x)$ be the ratio between minimum and maximum distance from x to the boundary $\delta(\sigma)$ of the object σ under L_2 norm. In other words, $\alpha(x) = \frac{\min_{y \in \delta(\sigma)} d(x,y)}{\max_{y \in \delta(\sigma)} d(x,y)}$, where $d(x,y)$ is the distance (under L_2 norm) between x and y (refer to Figure 3a). The *aspect ratio* $\alpha(\sigma)$ of an object σ is defined as the maximum value of $\alpha(x)$ for any point $x \in \sigma$, i.e., $\alpha(\sigma) = \max\{\alpha(x) : x \in \sigma\}$. An object is said to be α -*aspect object* if its aspect ratio is α . A point $c \in \sigma$ with $\alpha(c) = \alpha(\sigma)$ is defined as the *center* of the object σ (see Figure 3a). The minimum (resp., maximum) distance (under L_2 norm) from the center to the boundary of the object is referred to as the *width* (resp., *height*) of the object. Note that for any object σ , we have $0 < \alpha(\sigma) \leq 1$. The maximum possible value of α is attained when the object is ball.

Observe following properties of an α -aspect object.

- **Property 1.** *Let $\sigma \subseteq \mathbb{R}^d$ be an α -aspect object, then*
 - *any arbitrary rotated object σ' of σ is an α -aspect object;*
 - *any translated object σ' of σ is an α -aspect object;*
 - *the scaled object $\lambda\sigma$ is an α -aspect object, where $\lambda \in \mathbb{R}^+$.*



■ **Figure 3** Geometric interpretation of (a) aspect ratio and (b) aspect_∞ ratio.

- **Property 2.** Let $\sigma \subseteq \mathbb{R}^d$ be an α -aspect object centered at a point c with width w , then
- a ball of radius w centered at c is completely contained in σ , and
 - a ball of radius $\frac{w}{\alpha}$ centered at c contains the object σ .

Considering L_∞ norm instead of L_2 norm, similar to the above, one can define the aspect_∞ ratio, center, width and height of an object. An object with aspect_∞ ratio α is said to be an α -aspect_∞ object. Note that for any object σ , the value of $\alpha_\infty(\sigma)$ is also greater than zero and the maximum possible value of $\alpha_\infty(\sigma)$ is one which is attained for the hypercube. Analogous to Property 1 and 2, we have the following.

- **Property 3.** Let $\sigma \subseteq \mathbb{R}^d$ be an α -aspect_∞ object, then
- any translated object σ' of σ is an α -aspect_∞ object;
 - the scaled object $\lambda\sigma$ is an α -aspect_∞ object, where $\lambda \in \mathbb{R}^+$.
- **Property 4.** Let $\sigma \subseteq \mathbb{R}^d$ be an α -aspect_∞ object centered at a point c with width w , then
- a hypercube of side length $2w$ centered at c is completely contained in σ , and
 - a hypercube of side length $\frac{2w}{\alpha}$ centered at c contains the object σ .

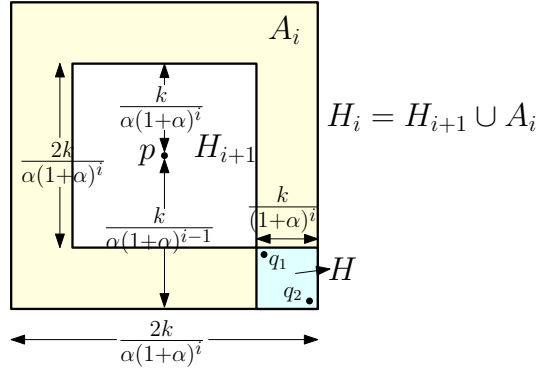
3 Upper Bound for Bounded Scaled Objects

3.1 For Objects with Fixed Aspect_∞ Ratio

In this section, we present an upper bound of the competitive ratio for piercing objects with a fixed α -aspect_∞ ratio. In the proof of the following theorem, all the distances, if explicitly not mentioned, are under L_∞ norm, and all the hypercubes are axis-parallel.

- **Theorem 7.** For piercing α -aspect_∞ objects in \mathbb{R}^d having width in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $\left(\frac{2}{\alpha}\right)^d ((1 + \alpha)^d - 1) \log_{(1+\alpha)}\left(\frac{2k}{\alpha}\right) + 1$.

Proof. Let \mathcal{I} be the set of input α -aspect_∞ objects presented to the algorithm. Let \mathcal{A} and \mathcal{O} be the piercing set returned by ALGORITHM-CENTER and the offline optimal, respectively, for \mathcal{I} . Let $p \in \mathcal{O}$ be a piercing point and let $\mathcal{I}_p \subseteq \mathcal{I}$ be the set of input α -aspect_∞ objects that



■ **Figure 4** Illustration.

are pierced by the point p . Let $\mathcal{A}_p \subseteq \mathcal{A}$ be the set of piercing points placed by ALGORITHM-CENTER to pierce all the input objects in \mathcal{I}_p . It is easy to see that $\mathcal{A} = \cup_{p \in \mathcal{O}} \mathcal{A}_p$. Therefore, the competitive ratio of our algorithm is upper bounded by $\max_{p \in \mathcal{O}} |\mathcal{A}_p|$.

Let us consider any point $a \in \mathcal{A}_p$. Since a is the center of an α -aspect $_\infty$ object $\sigma \in \mathcal{I}_p$ containing the point p having width at most k , the distance (under L_∞ norm) between a and p is at most $\frac{k}{\alpha}$. Therefore, a hypercube H_1 of side length $\frac{2k}{\alpha}$, centered at p , contains all the points in \mathcal{A}_p . Let H_i be a hypercube centered at p having side length $\frac{2k}{\alpha(1+\alpha)^{i-1}}$, where $i \in [m]$ and m is the smallest integer such that $\frac{2k}{\alpha(1+\alpha)^{m-1}} \leq 1$. Note that H_1, H_2, \dots, H_m are concentric hypercubes. Let us define the annular region $A_i = H_i \setminus H_{i+1}$, where $i \in [m-1]$. Let $\mathcal{A}_{p,m} = \mathcal{A}_p \cap H_m$, and $\mathcal{A}_{p,i} = \mathcal{A}_p \cap A_i$ be the subset of \mathcal{A}_p that is contained in the region A_i , for $i \in [m-1]$. Since the distance (under L_∞ norm) between any two points in H_m is at most one, and the width of any object in \mathcal{I}_p is at least one; therefore, any object belonging to \mathcal{I}_p having center in H_m contains the entire hypercube H_m . As a result, our online algorithm places at most one piercing point in H_m . Thus, $|\mathcal{A}_{p,m}| \leq 1$.

► **Lemma 8.** $|\mathcal{A}_{p,i}| \leq 2^d \left(\left(1 + \frac{1}{\alpha}\right)^d - \left(\frac{1}{\alpha}\right)^d \right)$, where $i \in [m-1]$.

Proof. Since $A_i = H_i \setminus H_{i+1}$, the distance (under L_∞ norm) from the center p to the boundary of H_i and H_{i+1} is $\frac{k}{\alpha(1+\alpha)^{i-1}}$ and $\frac{k}{\alpha(1+\alpha)^i}$, respectively. So the annular region A_i can contain hypercubes of side length $\frac{k}{(1+\alpha)^i}$. It is easy to observe that, the annular region A_i is the union of at most $2^d \left(\left(1 + \frac{1}{\alpha}\right)^d - \left(\frac{1}{\alpha}\right)^d \right)$ disjoint hypercubes, each having side length $\frac{k}{(1+\alpha)^i}$. To complete the proof, next, we argue that our online algorithm places at the most one piercing point in each of these hypercubes to pierce the objects in \mathcal{I}_p . Let H be any such hypercube of side length $\frac{k}{(1+\alpha)^i}$, and let $q_1 \in H$ be a piercing point placed by our online algorithm. For a contradiction, let us assume that our online algorithm places another piercing point $q_2 \in H$, where q_2 is the center of an object $\sigma \in \mathcal{I}_p$. Since σ contains both the points p and q_2 , and the distance (under L_∞ norm) between them is at least $\frac{k}{\alpha(1+\alpha)^i}$, therefore, the height of the object σ is at least $\frac{k}{\alpha(1+\alpha)^i}$ and the width is at least $\frac{k}{(1+\alpha)^i}$. Note that the distance (under L_∞ norm) between any two points in H is at most $\frac{k}{(1+\alpha)^i}$, as a result, σ is already pierced by q_1 . This contradicts our algorithm. Thus, the region H contains at most one piercing point of $\mathcal{A}_{p,i}$. Hence, the lemma follows. ◀

Since $\cup \mathcal{A}_{p,i} = \mathcal{A}_p$, we have $|\mathcal{A}_p| \leq 2^d \left(\left(1 + \frac{1}{\alpha}\right)^d - \left(\frac{1}{\alpha}\right)^d \right) (m-1) + 1$. As the value of $m \leq \log_{(1+\alpha)} \left(\frac{2k}{\alpha} \right) + 1$, the theorem follows. ◀

When objects are fixed oriented hypercubes and congruent balls in \mathbb{R}^d , then the value of α is 1 and $\frac{1}{\sqrt{d}}$, respectively. As a result, we have the following.

► **Corollary 9.** For piercing axis-aligned d -dimensional hypercubes with side length in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $2^d (2^d - 1) \log_2(2k) + 1$.

► **Corollary 10.** For piercing d -dimensional balls with radius in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $(2\sqrt{d})^d \left(1 + \frac{1}{\sqrt{d}}\right)^d - 1 \log_{(1+\frac{1}{\sqrt{d}})}(2k\sqrt{d}) + 1$.

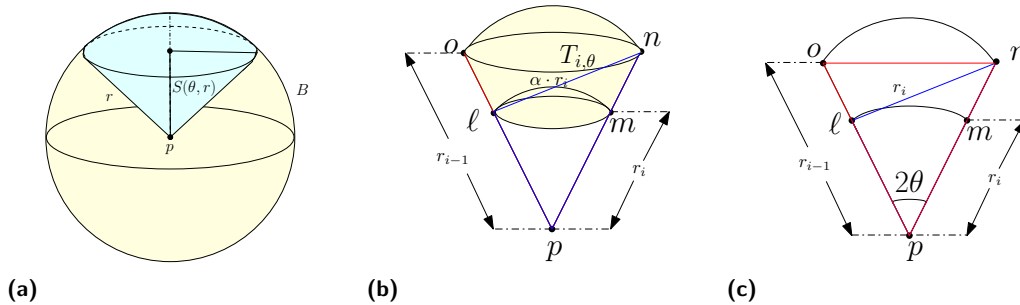
Unit Covering Problem in \mathbb{R}^d

In Theorem 7, if we place the value of $k = 1$, then due to Theorem 6, we have following result for the unit covering problem.

► **Corollary 11.** For the unit covering problem in \mathbb{R}^d , there exists a deterministic online algorithm whose competitive ratio is at most $\left(\frac{2}{\alpha}\right)^d ((1 + \alpha)^d - 1) \log_{(1+\alpha)}\left(\frac{2}{\alpha}\right) + 1$, where α is the aspect_∞ ratio of the unit object.

3.2 For α -Aspect Objects in \mathbb{R}^3

► **Theorem 12.** For piercing α -aspect object in \mathbb{R}^3 having width in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $\frac{2}{1-\cos(\theta)} \lceil \log_{(1+x)}\left(\frac{2k}{\alpha}\right) \rceil + 1$, where $\theta = \frac{1}{2} \cos^{-1}\left(\frac{1}{2} + \frac{1}{1+\sqrt{1+4\alpha^2}}\right)$ and $x = \frac{\sqrt{1+4\alpha^2}-1}{2}$.



■ **Figure 5** (a) Partitioning the ball B of radius r using spherical sectors $S(\theta, r)$; (b) Description of spherical sector $S(\theta, r_{i-1})$ and spherical block $T_{i, \theta}$; (c) Projection of spherical sector $S(\theta, r_{i-1})$.

Proof. Let \mathcal{I} be the set of input α -aspect objects in \mathbb{R}^3 presented to the algorithm. Let \mathcal{A} and \mathcal{O} be two piercing sets for \mathcal{I} returned by ALGORITHM-CENTER and the offline optimal, respectively. Let p be any piercing point of the offline optima \mathcal{O} . Let $\mathcal{I}_p \subseteq \mathcal{I}$ be the set of input objects pierced by the point p . Let \mathcal{A}_p be the set of piercing points placed by our algorithm to pierce all the objects in \mathcal{I}_p . To prove the theorem, we will give an upper bound of $|\mathcal{A}_p|$.

Let us consider any point $a \in \mathcal{A}_p$. Since a is the center of an α -aspect object $\sigma \in \mathcal{I}_p$ containing the point p , with width at most k (height at most $\frac{k}{\alpha}$), the distance between a and p is at most $\frac{k}{\alpha}$. Therefore, a ball B_1 of radius $\frac{k}{\alpha}$, centered at p , contains all the points in \mathcal{A}_p . Let $x = \frac{\sqrt{1+4\alpha^2}-1}{2}$ be a positive constant. Let B_i be a ball centered at p having radius $r_i = \frac{k}{\alpha(1+x)^{i-1}}$, where $i \in [m]$ and m is the smallest integer such that $\frac{k}{\alpha(1+x)^{m-1}} \leq \frac{1}{2}$. Note that B_1, B_2, \dots, B_m are concentric balls, centered at p . Let $\theta = \frac{1}{2} \cos^{-1}\left(\frac{1}{2} + \frac{1}{1+\sqrt{1+4\alpha^2}}\right)$ be

17:10 Online Piercing of Geometric Objects

a constant angle in $(0, \frac{\pi}{10}]$. Let $S(\theta, i)$ be a *spherical sector* obtained by taking the portion of the ball B_i by a conical boundary with apex at the center p of the ball and θ as the half of the cone angle (for an illustration see figure 5a). For any $i \in [m - 1]$, let us define the i th spherical block $T_{i,\theta} = S(\theta, i) \setminus S(\theta, i + 1)$.

▷ **Claim 13.** The distance between any two points in $T_{i,\theta}$ is at most αr_i .

Proof. Observe Figure 5b, where a detail of a spherical block is depicted. Note that the maximum distance between any two points in $T_{i,\theta}$ is at most $\max\{\overline{ln}, \overline{on}\}$. First, consider the triangle $\triangle lpn$ (see Figure 5b and 5c). By cosine rule of triangle, we have:

$$\begin{aligned} \overline{ln}^2 &= \overline{pl}^2 + \overline{pn}^2 - 2\overline{pl}\overline{pn}\cos(2\theta) \\ &= \left(\frac{k}{\alpha(1+x)^{i-1}}\right)^2 + \left(\frac{k}{\alpha(1+x)^{i-2}}\right)^2 - 2\left(\frac{k}{\alpha(1+x)^{i-1}}\right)\left(\frac{k}{\alpha(1+x)^{i-2}}\right)\cos(2\theta) \\ &= \left(\frac{k}{\alpha(1+x)^{i-2}}\right)^2 \left(\left(\frac{1}{(1+x)}\right)^2 + 1 - 2\left(\frac{1}{(1+x)}\cos(2\theta)\right) \right) \\ &= \left(\frac{k}{\alpha(1+x)^{i-1}}\right)^2 (1 + (1+x)^2 - 2(1+x)\cos(2\theta)) \end{aligned}$$

Since $\theta = \frac{1}{2}\cos^{-1}\left(\frac{1}{2} + \frac{1}{1+\sqrt{1+4\alpha^2}}\right)$, and $x = \frac{\sqrt{1+4\alpha^2}-1}{2}$, therefore, $\cos(2\theta) = \frac{(x+2)}{2(x+1)}$ and $x^2 + x = \alpha^2$. Now substituting these values in the above equation, we get

$$\overline{ln}^2 = r_i^2 (1 + (1+x)^2 - (2+x)) = r_i^2 (1 + 1 + x^2 + 2x - 2 - 2x) = r_i^2 (x^2 + x) = (\alpha r_i)^2.$$

Now, consider the triangle $\triangle opn$ (see Figure 5b and 5c). Here we have:

$$\begin{aligned} \overline{on}^2 &= 2\left(\frac{k}{\alpha(1+x)^{i-2}}\right)^2 - 2\left(\frac{k}{\alpha(1+x)^{i-2}}\right)^2\cos(2\theta) = 2\left(\frac{k}{\alpha(1+x)^{i-2}}\right)^2(1 - \cos(2\theta)) \\ &= 2\left(\frac{k}{\alpha(1+x)^{i-1}}\right)^2(1+x)^2(1 - \cos(2\theta)) = 2r_i^2(1+x)^2(1 - \cos(2\theta)). \end{aligned}$$

Now substituting the values of $\cos(2\theta) = \frac{(x+2)}{2(x+1)}$ and $x^2 + x = \alpha^2$ in the above equation, we get

$$\begin{aligned} \overline{on}^2 &= 2r_i^2(1+x)^2\left(1 - \frac{(x+2)}{2(x+1)}\right) = 2r_i^2(1+x)^2\left(\frac{2(x+1) - (x+2)}{2(x+1)}\right) \\ &= r_i^2(1+x)x = (\alpha r_i)^2. \end{aligned}$$

Note that $\overline{ln} = \overline{on} = \alpha r_i$, thus, αr_i is the maximum distance between any two points in the region $T_{i,\theta}$. ◁

► **Lemma 14.** For each $i \in [m]$, our algorithm places at most one piercing point in the spherical block $T_{i,\theta}$ to pierce any object in \mathcal{I}_p .

Proof. Let q_1 be the first piercing point placed by ALGORITHM-CENTER in $T_{i,\theta}$. For a contradiction, let us assume that ALGORITHM-CENTER places another piercing point $q_2 \in T_{i,\theta}$, where q_2 is center of some object $\sigma \in \mathcal{I}_p$. Since σ contains both points p and q_2 , and the distance between them is at least r_i , therefore, the height and width of σ are at least r_i and αr_i , respectively. Due to Claim 13, the distance between any two points in the spherical block $T_{i,\theta}$ is at most αr_i , as a result, σ is already pierced by q_1 . This contradicts that our algorithm places two piercing points in $T_{i,\theta}$. Hence, ALGORITHM-CENTER places at the most one piercing point in the spherical block $T_{i,\theta}$ to pierce objects in \mathcal{I}_p . ◁

Since the solid angle of the spherical sector $S(\theta, 1)$ is $2\pi(1-\cos(\theta))$ steradians, and the solid angle of a ball, measured from any point in its interior, is 4π steradians, therefore, we need at most $\frac{2}{1-\cos\theta}$ spherical sectors to cover the ball B_1 [16, §4.8.3]. Combining this with Lemma 14, we have $|\mathcal{A}_p| \leq \frac{2m}{(1-\cos\theta)}$. We can give a slightly better estimation, as follow. Since the distance between any two points in the innermost ball B_m is at most one, therefore any object in \mathcal{I}_p having center $q \in B_m$ contains the entire ball B_m . As a result, our online algorithm places at most one piercing point in B_m . Thus, $|\mathcal{A}_p| \leq \frac{2(m-1)}{1-\cos(\theta)} + 1 \leq \frac{2(\log_{(1+x)}(\frac{2k}{\alpha}))}{(1-\cos(\theta))} + 1$. This completes the proof of the theorem. ◀

As a consequence of Property 1 and Theorem 12, we have the following.

► **Corollary 15.** *For piercing arbitrary oriented scaled copies of an object in \mathbb{R}^3 with aspect ratio α and having width in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $\frac{2}{1-\cos(\theta)} \lceil \log_{(1+x)}(\frac{2k}{\alpha}) \rceil + 1$, where $\theta = \frac{1}{2} \cos^{-1} \left(\frac{1}{2} + \frac{1}{1+\sqrt{1+4\alpha^2}} \right)$ and $x = \frac{\sqrt{1+4\alpha^2}-1}{2}$.*

Analogous to Theorem 12, we have similar result for α -aspect objects in \mathbb{R}^2 . Here, we consider circular sector $C(\theta, r)$ with central angle θ instead of spherical sector $S(\theta, r)$ with central angle 2θ .

► **Theorem 16.** *For piercing α -aspect objects in \mathbb{R}^2 having width in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $\frac{2\pi}{\theta} (\log_{(1+x)}(2k/\alpha)) + 1$, where $\theta = \cos^{-1} \left(\frac{1}{2} + \frac{1}{1+\sqrt{1+4\alpha^2}} \right)$ and $x = \frac{\sqrt{1+4\alpha^2}-1}{2}$.*

Observe that for hypercubes in \mathbb{R}^d , the value of α is $\frac{1}{\sqrt{d}}$. As a result, we have the upper bound as follows.

► **Corollary 17.** *For piercing arbitrary oriented hypercubes in \mathbb{R}^3 (respectively, \mathbb{R}^2) with a side-length in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $223.98 \lceil \log_2(2\sqrt{3}k) \rceil + 1$ (respectively, $26.67 \lceil \log_2(2\sqrt{2}k) \rceil + 1$).*

Observe that for balls in \mathbb{R}^2 and \mathbb{R}^3 , the value of α is one. As a result, for $d = 2$ and 3 , we have a better estimation of the upper bound than Corollary 10. The result is as follows

► **Corollary 18.** *For piercing balls in \mathbb{R}^3 (respectively, \mathbb{R}^2) with a radius in the range $[1, k]$, ALGORITHM-CENTER achieves a competitive ratio of at most $58.861 \lceil \log_2(2k) \rceil + 1$ (respectively, $14.4 \lceil \log_2(2k) \rceil + 1$).*

4 Lower Bound for Bounded Scaled Objects

To obtain a lower bound, we think of a game between two players: Alice and Bob. Here, Alice plays the role of an adversary, and Bob plays the role of an online algorithm. In each round of the game, Alice presents an object such that Bob needs to place a new piercing point, i.e., the object does not contain any of the previously placed piercing points. To obtain a lower bound of the competitive ratio of $\Omega(z)$, it is enough to show that Alice can present a sequence of z nested objects in a sequence of z consecutive rounds of the game such that an offline optimum algorithm uses only one point to pierce all of these objects. First, we consider when objects are α -aspect objects in \mathbb{R}^2 , followed by axis parallel hypercubes in \mathbb{R}^d .

4.1 α -Aspect Objects in \mathbb{R}^2

► **Theorem 19.** *The competitive ratio of every deterministic online algorithm for piercing α -aspect objects in \mathbb{R}^2 having width in the range $[1, k]$ is at least $\log_{(\frac{2}{\alpha})} k$ for $k > \frac{2}{\alpha}$.*

Proof. To prove the lower bound, we adaptively construct a sequence of $m = \log_{(\frac{2}{\alpha})} k$ objects each with aspect ratio α and width in the range $[1, k]$ such that any online algorithm needs at least m piercing points to pierce them; while the offline optimal needs just one point. In each round of the game, the adversary presents an object of width in the range $[1, k]$, and the online algorithm needs to place a piercing point to pierce the presented object if it is already not pierced by any of the previously placed piercing points. Let $w(\sigma)$ and $h(\sigma)$ be width and height of an object σ , respectively. Let σ_1 , having width $w(\sigma_1) = k$, be the first input object presented to the algorithm. For the sake of simplicity, let us assume that the center c_1 of σ_1 coincides with the origin. All remaining objects $\sigma_2, \sigma_3, \dots, \sigma_m$ are presented adaptively depending on the position of the piercing points p_1, p_2, \dots, p_{m-1} placed by the algorithm. For $i = 1, 2, \dots, m = \log_{(\frac{2}{\alpha})} k$, we maintain the following two invariants.

- (1) The object σ_i having width $k \left(\frac{\alpha}{2+\epsilon}\right)^{i-1}$ is not pierced by any of the previously placed piercing point p_j , where $j < i$.
- (2) The object σ_i is totally contained in the object σ_{i-1} .

Invariant (1) ensures that any online algorithm needs $m+1$ piercing points; while invariant (2) ensures that all the objects $\sigma_1, \sigma_2, \dots, \sigma_m$ can be pierced by a single point. For $i = 1$, both invariants hold trivially. At the beginning of round $i = 1, 2, \dots, m-1$, assume that both invariants hold. Depending on the position of the previously placed piercing point p_i , in the $(i+1)$ th round of the game, the object σ_{i+1} , having width $w(\sigma_{i+1}) = k \left(\frac{\alpha}{2+\epsilon}\right)^i$ ($h(\sigma_{i+1}) = \frac{k}{2+\epsilon} \left(\frac{\alpha}{2+\epsilon}\right)^{i-1}$), is presented to the algorithm. Here $\epsilon > 0$ is an arbitrary constant very close to zero. The center c_{i+1} of σ_{i+1} is defined as the following.

$$c_{i+1} = \begin{cases} c_i + \frac{w(\sigma_i)}{2}, & \text{if } p_i(x_1) \leq c_i(x_1), \\ c_i - \frac{w(\sigma_i)}{2}, & \text{otherwise (i.e., } p_i(x_1) > c_i(x_1)), \end{cases}$$

where $p_i(x_1)$ and $c_i(x_1)$ denotes the first coordinate of p_i and c_i , respectively.

First, we show that σ_{i+1} is totally contained in σ_i . Observe that, depending on the position of p_i , the center of σ_{i+1} is either $c_i + \frac{w(\sigma_i)}{2}$ or $c_i - \frac{w(\sigma_i)}{2}$. In both cases, we have $\text{dist}(c_i, c_{i+1}) = \frac{w(\sigma_i)}{2}$. On the other hand, $h(\sigma_{i+1}) = \frac{k}{2+\epsilon} \left(\frac{\alpha}{2+\epsilon}\right)^{i-1} < \frac{k}{2} \left(\frac{\alpha}{2+\epsilon}\right)^{i-1} = \frac{w(\sigma_i)}{2}$. Hence, σ_{i+1} is totally contained in σ_i . Thus, invariant (2) is maintained.

Note that $\text{dist}(p_i, c_{i+1})$ is greater than the height of σ_{i+1} since

$$\text{dist}(p_i, c_{i+1}) > \frac{w(\sigma_i)}{2} = \frac{k}{2} \left(\frac{\alpha}{2+\epsilon}\right)^{i-1} > \frac{k}{2+\epsilon} \left(\frac{\alpha}{2+\epsilon}\right)^{i-1} = h(\sigma_{i+1}).$$

The first inequality follows from the definition of c_{i+1} . Thus, σ_{i+1} does not contain the point p_i . Due to induction hypothesis, σ_i does not contain any of the previously placed piercing point p_j for $j < i$ and from invariant (2) we know that σ_{i+1} is contained in σ_i . Hence, σ_{i+1} does not contain any of the previously placed piercing point p_j , for $j < i+1$. Thus, invariant (1) is maintained.

Note that when $m > \log_{\frac{2+\epsilon}{\alpha}} k + 1$, the width of the object σ_m , i.e., $k \left(\frac{\alpha}{2+\epsilon}\right)^{m-1}$ is less than one. In other words, for any $m \leq \log_{\frac{2+\epsilon}{\alpha}} k + 1$, we can construct the input sequence satisfying both invariants. Since $\epsilon > 0$ is arbitrarily constant very close to 0, we choose the value of $m = \log_{\frac{2}{\alpha}} k$. Hence, the theorem follows. ◀

4.2 Fixed Oriented Hypercubes in \mathbb{R}^d

In this subsection, all the hypercubes are axis parallel. First, we establish the following essential ingredient to prove our main result.

► **Lemma 20.** *GAME-OF-SAME-SIDE(r) ($GSS(r)$):*

For any $r \in \mathbb{R}^+$, in a consecutive d rounds of the game, Alice can adaptively present d hypercubes $\sigma_1, \sigma_2, \dots, \sigma_d$, each having side length r , such that

- (i) *Bob needs to place d points to pierce them;*
- (ii) *the common intersection region $Q = \cap_{i=1}^d \sigma_i$ is nonempty;*
- (iii) *moreover, Q contains an empty hypercube E , of side length at least $\frac{r}{2} - \epsilon$, not containing any of the d piercing points placed by Bob. Here $\epsilon > 0$ is a very small constant close to 0.*

We refer a consecutive d rounds of the game satisfying the above lemma as a $GSS(r)$. The first hypercube σ_1 is denoted as the *starter* and the hypercube E as the *empty hypercube* since it does not contain any of the piercing points placed by Bob in the $GSS(r)$. Note that any hypercube of side length r could be the starter of this d round game.

Proof. Let σ_1 be a hypercube of side length r presented by Alice in the first round of the game. Throughout the proof of this claim, for the sake of simplicity, we assume that the center of σ_1 is the origin (if not, we can always translate the coordinate system to make it). Alice presents the remaining hypercubes $\sigma_2, \dots, \sigma_d$ adaptively depending on Bob's moves. We maintain the following two invariants: For $i = 1, \dots, d$, when Alice presents hypercubes $\sigma_1, \dots, \sigma_i$, each of side length r , and Bob presents piercing points p_1, \dots, p_i ,

- (I) the hypercube σ_i is not pierced by any of the previously placed piercing point p_j , where $j < i$;
- (II) the common intersection region $Q_i = \cap_{j=1}^i \sigma_j$ is a hyperrectangle whose first $(i - 1)$ sides are of length $\frac{r}{2} - \epsilon$ each, and each of the remaining sides are of length r . Moreover, Q_i does not contain any of the points p_j , where $j < i$.

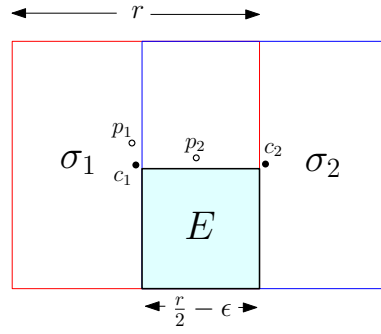
An illustration of the planar version of the game appears in Figure 6. For $i = 1$, both the invariants trivially hold. At the beginning of round i (for $i = 2, \dots, d$), assume that both invariants hold. Let us define a translation vector $\mathbf{v}_i \in \mathbb{R}^d$ as $\mathbf{v}_i = (s(1) (\frac{r}{2} + \epsilon), s(2) (\frac{r}{2} + \epsilon), \dots, s(i - 1) (\frac{r}{2} + \epsilon), 0, \dots, 0)$, where for any $j < i$, we have

$$s(j) = \begin{cases} +1, & \text{if } p_j(x_j) < 0, \text{ where } p_j(x_j) \text{ is } j\text{th coordinate of } p_j, \\ -1, & \text{otherwise.} \end{cases}$$

We define $\sigma_i = \sigma_1 + \mathbf{v}_i$. For any $j < i$, due to the definition of the j th component of the translation vector \mathbf{v}_i , the hypercube σ_i does not contain the point p_j . Hence, invariant (I) is maintained.

Note that $\sigma_1 \cap \sigma_i$ is a hyperrectangle whose first $(i - 1)$ sides are of length $\frac{r}{2} - \epsilon$ each, and each of the remaining sides are of length r . On the other hand, from the assumption, we know that Q_{i-1} is a hyperrectangle whose first $(i - 2)$ sides are of length $\frac{r}{2} - \epsilon$ each, and each of the remaining sides are of length r . Therefore, $Q_i = Q_{i-1} \cap \sigma_i$ is a hyperrectangle whose first $(i - 1)$ sides are of length $\frac{r}{2} - \epsilon$ each, and each of the remaining sides are of length r . Since σ_i does not contain any of the points p_j where $j < i$, as a result the hyperrectangle $Q_i \subseteq \sigma_i$ also does not contain any of them. Therefore, invariant (II) is also maintained.

17:14 Online Piercing of Geometric Objects



■ **Figure 6** Illustration of the GAME-OF-SAME-SIDE(r) for $d = 2$. Initially Alice presented hypercube σ_1 of side length r in the first round of the game. Then, Alice presents the second hypercube σ_2 adaptively, depending on Bob's moves, such that $p_1 \notin \sigma_2$. Also, the common intersection region $\cap_{j=1}^2 \sigma_j$ is a rectangle whose sides are of length $\frac{r}{2} - \epsilon$ and r . Moreover, the intersection region contains an *empty* hypercube E , of side length at least $\frac{r}{2} - \epsilon$, not containing p_1 and p_2 . Here $\epsilon > 0$ is a very small constant close to 0.

At the end of the d th round of the game, we have the hyperrectangle $Q = Q_d$ (whose d th side is only of length r , other sides are of length $\frac{r}{2} - \epsilon$ each) that does not contain any of the points p_j where $j < d$, but it may contain the point p_d . Depending on the d th coordinate of the point p_d , we can construct a hypercube $E \subset Q$ of length $\frac{r}{2} - \epsilon$ that does not contain the point p_d . Hence, the lemma follows. ◀

Now, we prove the following main theorem.

► **Theorem 21.** *The competitive ratio of every deterministic online algorithm for piercing fixed oriented hypercubes in \mathbb{R}^d with side length in the range $[1, k]$ is at least $d \left(\log_2 \left(\frac{k}{1+2\epsilon} \right) \right) + 2^d$.*

Proof. To prove the main theorem, we maintain the following two invariants: For $i = 1, \dots, z = d \left(\left(\frac{k}{1+2\epsilon} \right) - 1 \right) + 2^d$, when Alice presents hypercubes $\sigma_1, \dots, \sigma_i$ and Bob presents piercing points p_1, \dots, p_i ,

(III) The hypercube σ_i is not pierced by any of the previously placed piercing point p_j , where $j < i$.

(IV) The intersection region $\cap_{j=1}^i \sigma_j$ is nonempty.

Invariant (III) implies that Bob is forced to place z piercing points, while Invariant (IV) ensures that all the hypercubes $\sigma_1, \sigma_2, \dots, \sigma_z$ can be pierced by a single piercing point.

We evoke the $GSS(r)$ for $t > \left(\log_2 \left(\frac{k}{1+2\epsilon} \right) - 1 \right)$ times with different values of $r = k, \frac{k}{2}, \frac{k}{4}, \dots, \frac{k}{2^t} - \frac{2^t - 1}{2^t - 1} \epsilon$. More specifically, the empty hypercube $E_{t'}$ generated by $GSS\left(\frac{k}{2^{t'-1}} - \frac{2^{t'} - 1}{2^{t'} - 1} \epsilon\right)$ plays the role of the starter hypercube for the next evocation of $GSS\left(\frac{k}{2^{t'}} - \frac{2^{t'} - 1}{2^{t'} - 1} \epsilon\right)$, where $1 \leq t' < t$. Since the empty hypercube of t' th game plays the role of the starter of $(t' + 1)$ th game, it is straightforward to see that the set of td hypercubes $\sigma_1, \sigma_2, \dots, \sigma_{td}$ and set of points p_1, p_2, \dots, p_{td} satisfies both invariants (III) and (IV). At the end of the td th GSS , the empty hypercube E_{td} has side length at least 1. Due to Corollary 3, starting with the hypercube E_{td} , Alice can adaptively present 2^d (including E_{td}) hypercubes each of side length 1 such that both invariants (III) and (IV) are maintained. This completes the proof of the theorem. ◀

4.3 Balls in \mathbb{R}^2 and \mathbb{R}^3

Similar to the proof of Theorem 21, one can prove the following lower bound result for balls in \mathbb{R}^3 (respectively, \mathbb{R}^2). Also, note that we could not generalize this to higher dimensional balls.

► **Theorem 22.** *The competitive ratio of every deterministic online algorithm for piercing balls in \mathbb{R}^3 (respectively, \mathbb{R}^2) with radius in the range $[1, k]$ is at least $3 \log_4 k + 1$ (respectively, $2 \log_{(\frac{8}{3})} k + 1$).*

5 Conclusion

We show that no online algorithm can obtain a competitive ratio lower than $\Omega(n)$ for piercing n intervals. Due to this pessimistic result, we restricted our attention to special kinds of objects, i.e., bounded scaled objects. We propose upper bounds for piercing bounded scaled α -aspect $_\infty$ objects in higher dimension. By placing specific values of α , the tight asymptotic bounds can be obtained for bounded scaled objects (like intervals, squares, disks, and α -aspect polygons) in \mathbb{R} and \mathbb{R}^2 . For bounded scaled α -aspect $_\infty$ objects in \mathbb{R}^d , it is possible to generalize our lower bound result obtained for hypercubes. For higher dimensions, for example, for d -dimensional hypercubes, there is a huge gap between the lower and the upper bound. We propose bridging these gaps as a future direction of research. We only consider the deterministic model. This raises the question of whether randomization helps.

References

- 1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. doi:10.1137/060661946.
- 2 C. J. Argue, Anupam Gupta, Ziyi Tang, and Guru Guruganesh. Chasing convex bodies with linear competitive ratio. *J. ACM*, 68(5):32:1–32:10, 2021. doi:10.1145/3450349.
- 3 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- 4 Sébastien Bubeck, Bo’az Klartag, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Chasing nested convex bodies nearly optimally. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1496–1508. SIAM, 2020. doi:10.1137/1.9781611975994.91.
- 5 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 861–868. ACM, 2019. doi:10.1145/3313276.3316314.
- 6 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.
- 7 Timothy M. Chan and Hamid Zarrabi-Zadeh. A randomized algorithm for online unit clustering. *Theory Comput. Syst.*, 45(3):486–496, 2009. doi:10.1007/s00224-007-9085-7.
- 8 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. doi:10.1137/S0097539702418498.
- 9 Adrian Dumitrescu, Anirban Ghosh, and Csaba D. Tóth. Online unit covering in Euclidean space. *Theor. Comput. Sci.*, 809:218–230, 2020. doi:10.1016/j.tcs.2019.12.010.
- 10 Adrian Dumitrescu and Csaba D. Tóth. Online unit clustering and unit covering in higher dimensions. *Algorithmica*, 84(5):1213–1231, 2022. doi:10.1007/s00453-021-00916-6.

- 11 Alon Efrat, Matthew J. Katz, Frank Nielsen, and Micha Sharir. Dynamic data structures for fat objects and their applications. *Comput. Geom.*, 15(4):215–227, 2000. doi:10.1016/S0925-7721(99)00059-0.
- 12 Guy Even and Shakhar Smorodinsky. Hitting sets online and unique-max coloring. *Discret. Appl. Math.*, 178:71–82, 2014. doi:10.1016/j.dam.2014.06.019.
- 13 Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.
- 14 Joel Friedman and Nathan Linial. On convex body chasing. *Discret. Comput. Geom.*, 9:293–321, 1993. doi:10.1007/BF02189324.
- 15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 16 J. W. Harris and H. Stocker. *Handbook of Mathematics and Computational Science*. New York: Springer-Verlag, 1998.
- 17 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 18 Matthew J. Katz, Frank Nielsen, and Michael Segal. Maintenance of a piercing set for intervals with applications. *Algorithmica*, 36(1):59–73, 2003. doi:10.1007/s00453-002-1006-1.
- 19 Frank Nielsen. Fast stabbing of boxes in high dimensions. *Theor. Comput. Sci.*, 246(1-2):53–72, 2000. doi:10.1016/S0304-3975(98)00336-3.
- 20 János Pach, Peter Brass, and William Moser. Research problems in discrete geometry. *Research Problems in Discrete Geometry*, January 2005. doi:10.1007/0-387-29929-7.
- 21 Mark Sellke. Chasing convex bodies optimally. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1509–1518. SIAM, 2020. doi:10.1137/1.9781611975994.92.


Half-Guarding Weakly-Visible Polygons and Terrains

Nandhana Duraisamy ✉

PSG College of Technology, Coimbatore, India

Hannah Miller Hillberg ✉ 🏠

University of Wisconsin, Oshkosh, WI, USA

Ramesh K. Jallu¹ ✉ 

Indian Institute of Information Technology Raichur, India

Erik Krohn ✉ 🏠 

University of Wisconsin, Oshkosh, WI, USA

Anil Maheshwari ✉

Carleton University, Ottawa, Canada

Subhas C. Nandy ✉

Indian Statistical Institute, Kolkata, India

Alex Pahlow ✉

University of Wisconsin, Oshkosh, WI, USA

Abstract

We consider a variant of the art gallery problem where all guards are limited to seeing 180° . Guards that can only see in one direction are called half-guards. We give a polynomial time approximation scheme for vertex guarding the vertices of a weakly-visible polygon with half-guards. We extend this to vertex guarding the boundary of a weakly-visible polygon with half-guards. We also show NP-hardness for vertex guarding a weakly-visible polygon with half-guards. Lastly, we show that the orientation of half-guards is critical in terrain guarding. Depending on the orientation of the half-guards, the problem is either very easy (polynomial time solvable) or very hard (NP-hard).

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Art Gallery Problem, Approximation Algorithm, NP-Hardness, Monotone Polygons, Half-Guards

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.18

Related Version This paper combines the results of [7] and [12], which were obtained independently. The first gave a PTAS for vertex guarding the boundary of a weakly-visible polygon. The second gave a PTAS for vertex guarding the vertices of a weakly-visible polygon and provided hardness results for half-guarding terrains and weakly-visible polygons. A preliminary version of [12] is presented in the Iranian Conference on Computational Geometry 2022 (an informal meeting).

Previous Version: <https://iccg2022.ce.sharif.edu/files/Papers/4-Guarding-Weakly-Visible-Polygons-with-Half-Guards.pdf>

1 Introduction

An instance of the original *art gallery problem* takes as input a simple polygon P having a sequence of vertices $V = \{v_1, v_2, \dots, v_n\}$, and reports a guarding set of points $G \subseteq P$ such that every point $p \in P$ is seen by a point in G . Here, by the term *a point q sees a point p*

¹ The work was partially done while the author was affiliated with Indian Statistical Institute, Kolkata, India.



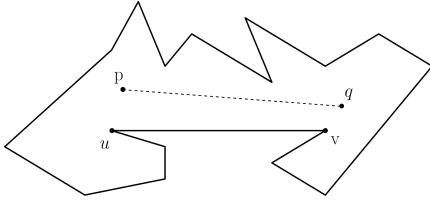
© Nandhana Duraisamy, Hannah Miller Hillberg, Ramesh K. Jallu, Erik Krohn, Anil Maheshwari, Subhas C. Nandy, and Alex Pahlow; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

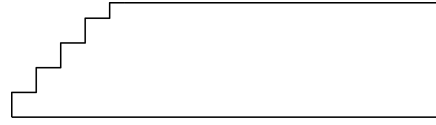
Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 18; pp. 18:1–18:17



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



■ **Figure 1** A WV-polygon where p sees q and every point in the polygon is seen by a point on $e = (u, v)$ or sees a point on e .



■ **Figure 2** A WV-polygon that requires $\Omega(n)$ half-guards that see to the right. Half-guards need to be placed at every step like structure.

where $p, q \in P$, we mean that the line segment $[p, q]$ completely lies inside the polygon P . In this paper, we study the vertex guarding problem which says that guards are only allowed to be placed at the vertices of V , and the objective is to find the smallest such G .

1.1 Definitions

We use $p.x$ to denote the x-coordinate for point p . A weakly-visible polygon (WV-polygon) P contains a visibility edge $e = (u, v)$ such that every point in P sees at least one point on the edge e . When referencing a half-guard p sees to the right (i.e., guards see 180° towards the right), then a point q is visible to p if $p.x \leq q.x$ (see Fig. 1). Without loss of generality, we assume that the visibility edge e is aligned with the x -axis.

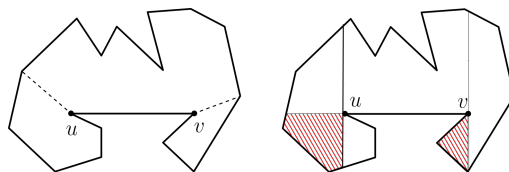
A (strictly) x -monotone terrain T with x -axis as the base is defined by a set of points $V = \{v_1, v_2, \dots, v_n\}$ satisfying $v_i.x < v_{i+1}.x$ for $i = 1, 2, \dots, n - 1$, and $[v_1.x, v_n.x]$ is aligned with the x -axis.

1.2 Previous Works

Extensive studies are done on algorithmic aspects of *art galleries*. Several results related to hardness and approximations are available in [1, 8, 11, 17]. Due to the inherent difficulty in fully understanding the art gallery problem for simple polygons, work has been done for guarding polygons with some special structures (see [2, 4, 15]). In this paper we consider terrains and weakly-visible polygons.

Motivation of considering these two special structure comes from the fact that many cameras/sensors cannot generally sense in 360° , referred to as *full-guards* in this paper. In [2], the authors proposed a constant-factor approximation algorithm to guard interior of a WV-polygon. We consider the *half-guards* that can sense in 180° . For weakly-visible polygons, we restrict the problem even further by only allowing these half-guards to see to the right. Even with these restrictions, the vertex guarding problem is difficult to solve in weakly-visible polygons. Using half-guarding the authors in [9, 14] propose a 4-approximation algorithm for terrain guarding with full-guards. In Fig. 2, we demonstrate that there are WV-polygons P that can be completely guarded with one full-guard, but requires $\Omega(n)$ half-guards.

The difficulty with guarding terrains with half-guards is less straightforward. As shown in [5], it is possible to optimally guard a terrain in polynomial time using half-guards that only see to the right. If the visibility of half-guards is modified such that they see down in the terrain setting, as we will show, the problem becomes NP-hard.



■ **Figure 3** On the left, a full-guard placed at u and v cuts off the polygon below the WV-edge. On the right, the shaded regions below \overline{uv} are still unseen after half-guards are placed at u and v .

1.3 Our Contribution

Ashur et al. [3] gives a local search based polynomial time approximation scheme (PTAS) for minimum dominating set in terrain-like graphs. They then show that several families of polygons have a visibility graph that is terrain-like. One such family is WV-polygons. However, their analysis uses the fact that vertices can see 360° and that the visibility graph of the vertices contains undirected edges. Since edges in a visibility graph for a WV-polygon that use half-guards are directed, it does not imply that the visibility graph is terrain-like. We provide additional observations in this paper which show a PTAS for vertex guarding the vertices of a WV-polygon with half-guards. We extend this to show that guarding the boundary of a WV-polygon with half-guards placed at vertices also admits a PTAS.

NP-hardness has been shown for many variants of the art gallery problem [4, 8, 17]. In many of those reductions, guards are allowed to see in all directions. If the problem is restricted enough, it can become polynomially time solvable, for example, see [6, 16]. If the polygon is restricted to be a WV-polygon, guards are restricted to be at the vertices and guards are only allowed to see to the right, even with these many restrictions, the problem is still NP-hard. To obtain this result, we first show that a variant of half-guarding a terrain is NP-hard.

In Section 2, we provide a PTAS for vertex guarding the vertices and boundary of a WV-polygon using half-guards that see right. In Section 3, we show that the half-guarding of terrains is NP-hard when the half-guards see down. In Section 4, the NP-hardness is shown for vertex guarding a WV-polygon using half-guards that see right.

2 PTAS for Vertex Guarding a WV-Polygon with Half-Guards

We use local search to obtain a PTAS for half-guarding a WV-polygon. We adapt the idea of the proof provided in [3] to this problem with some notable exceptions. The divergent details of the algorithm and proof along with a few identical details, included for readability, are provided below.

Much of the proof from [3] assumes that the WV-polygon lies above the WV-edge by placing guards at u and v and cutting off the portion of the polygon beneath the WV-edge, see Fig. 3(left). When every vertex under consideration lies above this WV-edge, the so-called order claim holds and the visibility graph is terrain-like. However, consider doing the same thing for a WV-polygon using half-guards that see to the right. In this case, the remaining unseen portion of the polygon cannot be assumed to be above the WV-edge. The shaded region in Fig. 3(right) demonstrates the invisible portions to the left-of-and-below the guards at u and v . A guard g *dominates* another guard g' if for every vertex v that g' sees, g also sees v . Unlike with full-guards, a half-guard placed at u will not dominate a half-guard placed in the shaded region to the left-of-and-below u . Thus, the proof of the order claim for WV-polygons with full-guards does not imply that the order claim is true for WV-polygons with half-guards.

■ **Algorithm 1** Local Search Algorithm.

We set $k = \frac{\alpha}{\epsilon^2}$, for appropriate constants $\alpha, \epsilon > 0$. Let Q be the set of guards.

1. Initialize Q to the vertex set V .
2. Determine if there exists a subset $S \subseteq Q$ of size at most k and a subset $S' \subseteq V \setminus Q$ of size at most $|S| - 1$ such that $(Q \setminus S) \cup S'$ guards V .
3. If such S and S' exists then, set $Q \leftarrow (Q \setminus S) \cup S'$ and go to step 2. Else, return the set Q .

2.1 Algorithm Overview

In this section, we will assume that $e = (u, v)$ is a convex edge, that is, the interior angles at u and v are convex. Without loss of generality, we assume that $P \setminus e$ is contained in the open half-plane above the x -axis. We later show how to solve for the general case, where the convexity assumption of the angles at u and v is removed. The local search algorithm, described below, is identical to the one described in [3].

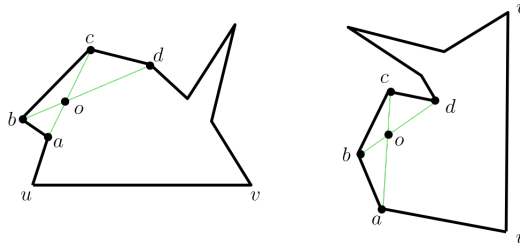
2.2 Local Search Analysis

For any two points a and b on the boundary of the polygon P , we use $a \prec b$ (or $b \succ a$) to say that a precedes b (or b succeeds a), i.e., when one reaches a before b , while traversing the boundary of P , clockwise from u . Claim 1 was proved to be true for weakly-visible polygons with full guards. The order claim proof from [3] shows that a and d see each other. However, they do not claim that d must be to the right of a . With full-guards, it does not matter since if a sees d , then d sees a . The following proof of the order claim for *half-guards* shows why a sees d . An important note to remember is that this claim is only true if the angles at u and v are convex.

▷ **Claim 1.** Let a, b, c, d be four points on the boundary of a WV-polygon such that $a \prec b \prec c \prec d$. If a sees c and b sees d , then a must see d (see Fig. 4).

Proof. Since a sees c and b sees d , $a.x \leq c.x$ and $b.x \leq d.x$. If $a.x \leq b.x$, then $a.x \leq b.x \leq d.x$. Using the proof from [3], a must see d . If $b.x < a.x$, then consider the line segment connecting b to d . Since $a \prec b \prec c \prec d$, it must be the case that the \overline{bd} line segment crosses the \overline{ac} line segment, see Fig. 4. Let o be the intersection point of \overline{ac} and \overline{bd} . By definition, $a.x \leq o.x$ and $o.x \leq d.x$. Therefore, $a.x \leq d.x$ and a sees d . ◁

It should be noted that the orientation of the WV-edge does not matter with Claim 1. In other words, if the polygon is rotated in any direction, the original order claim from [3] still holds, i.e. a and d must see each other, see Fig. 4(right).



■ **Figure 4** Order claim example. The orientation of the \overline{uv} edge does not matter. If $a \prec b \prec c \prec d$, a sees c and b sees d , then a sees d .

Let the red set R be an optimal solution (minimum cardinality guard set), and the blue set B be the solution returned by Algorithm 1. For each vertex $x \in (R \cup B)$, we define $\text{color}(x)$ to be the color of the vertex x , which is red if $x \in R$, and blue if $x \in B$. We can assume that $R \cap B = \emptyset$. Any vertices that are in both R and B are removed, i.e. the local search algorithm found an optimal guard and one can ignore vertices which are seen by such a guard. We prove that $|B| \leq (1 + \epsilon) \cdot |R|$. We construct a bipartite graph $G = (R \cup B, E)$. As in [3], we can prove that (i) G is planar and (ii) G satisfies the locality condition as defined below.

► **Definition 2.** A graph $G = (R \cup B, E)$ satisfies **locality condition** if for each vertex $w \in P$, there exist vertices $r \in R$ and $b \in B$, such that (i) r sees w , (ii) b sees w , & (iii) $(r, b) \in E$.

Next, by using the proof scheme of Mustafa and Ray [19], if G satisfies these two conditions, then we have $|B| \leq (1 + \epsilon) \cdot |R|$. As in [3], we now define $\lambda(\cdot)$ and $\rho(\cdot)$ for each vertex of P . While traversing the boundary clockwise from u , for a vertex $w \in V$, if there exists a vertex in $R \cup B$ that sees w and it precedes w on the path of traversal from u to w , then $\lambda(w)$ is set to the first such vertex. Similarly, $\rho(w)$ is set to the first vertex in $R \cup B$ that sees w while traversing the boundary counterclockwise from v . Since $R \cap B = \emptyset$, for every vertex $w \in V$, $\lambda(w)$ and/or $\rho(w)$ must be defined in $R \cup B$.

2.3 Constructing G

▷ **Claim 3.** Let $A_1 = \{(\lambda(w), w) \mid w \in V \text{ for which } \lambda(w) \text{ is defined}\}$. Then the segments of A_1 are non-crossing.

Proof. Consider two segments $(\lambda(x), x), (\lambda(y), y) \in A_1$, such that $\lambda(x) \neq \lambda(y)$. Without loss of generality assume that $\lambda(x) \prec \lambda(y)$. By contradiction, suppose the segments $(\lambda(x), x)$ and $(\lambda(y), y)$ intersect, then it must be the case where $\lambda(x) \prec \lambda(y) \prec x \prec y$. But, by Claim 1, this would imply that $\lambda(x)$ sees y , which is impossible by the definition of $\lambda(y)$. Thus, the segments are non-crossing. ◁

The edges of G are similar to the edges of G described in [3]. If it can be shown that the order claim holds and that the segments of A_1 are non-crossing, the details of how to create the graph are essentially the same. However, the description of those edges are provided here for completeness. For each vertex $x \in R \cup B$, if $\lambda(x)$ is defined and $\text{color}(\lambda(x)) \neq \text{color}(x)$, add the edge $(\lambda(x), x)$ to E_1 . Otherwise, if there exists a segment $(\lambda(w), w) \in A_1$, such that (i) $\lambda(w) \prec x \prec w$, (ii) $(\lambda(w), w)$ can be reached from x without going outside of P and without intersecting any other segment in A_1 (except possibly at x), and (iii) $\text{color}(\lambda(w)) \neq \text{color}(x)$, add the edge $(\lambda(w), x)$ to E_1 .

Analogously, we define the sets A_2 and E_2 by replacing λ with ρ . This implies, $A_2 = (w, \rho(w)) \mid w \in V \text{ for which } \rho(w) \text{ is defined}$, and E_2 is defined with respect to A_2 . For each vertex $w \in R \cup B$, if both $\lambda(x)$ and $\rho(x)$ are defined and $\text{color}(\lambda(x)) \neq \text{color}(\rho(x))$, then add the edge $(\lambda(x), \rho(x))$ to E_3 . Let $G = (R \cup B, E)$, where $E = E_1 \cup E_2 \cup E_3$. Now, we have the following lemmas.

► **Lemma 4.** The graph $G = (R \cup B, E)$ is (a) planar, and (b) satisfies locality condition.

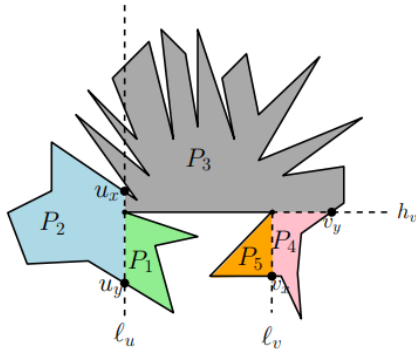
Proof. We prove that G is planar by finding a suitable embedding of it in the plane. Let C be a circle of radius $r > 0$ in the plane. We map the vertices of P to equally-spaced points on C , and the edges in E are drawn between pairs of points inside and outside of

C . If $A_1 \cap E_1 \neq \emptyset$, then the edges in $A_1 \cap E_1$ are drawn inside C as straight line segments, and are non-crossing due to Claim 3. Note that for each edge $(\lambda(w), x) \in E_1 \setminus A_1$, there exists a segment $(\lambda(w), w) \in A_1$ such that $\lambda(w) \prec x \prec w$ and x can reach $(\lambda(w), w)$ without exiting from P and without intersecting any segment in A_1 . We can argue that the edge $(\lambda(w), x)$ can be drawn as a straight line segment inside C without intersecting any edge that was already drawn. Suppose $(\lambda(w), x)$ intersects an edge (segment) $(\lambda(a), a) \in A_1 \cap E_1$. Now, if $\lambda(a) \prec \lambda(w) \prec a \prec x$, then the segments $(\lambda(a), a)$ and $(\lambda(w), w)$ are crossing, a contradiction to Claim 3. If $\lambda(w) \prec \lambda(a) \prec x \prec a$, there arise two cases: (i) $a \prec w$ or $a = w$, in this case x crosses the segment $(\lambda(a), a)$ to reach $(\lambda(w), w)$, a contradiction; (ii) $w \prec a$, in which case the segments $(\lambda(w), w)$ and $(\lambda(a), a)$ are crossing, a contradiction to Claim 3. If, suppose, $(\lambda(w), x)$ intersects an edge $(\lambda(w'), y) \in E_1 \setminus A_1$, then $\lambda(w) \prec \lambda(w') \prec x \prec y$. Implies $(\lambda(w), x)$ crosses the segment $(\lambda(w'), w') \in A_1$. If $(\lambda(w'), w') \in E_1$, then it is similar to the previous case. On the other hand, if $(\lambda(w'), w') \in A_1$, there arise two cases: either $w' \prec w$ or $w' = w$, in which the contradiction is that x crosses $(\lambda(w'), w')$ to reach $(\lambda(w), w)$; or $w \prec w'$, in which $\lambda(w)$ sees w' (order claim applied to $\lambda(w), \lambda(w'), w, w'$), contradiction to the definition of $\lambda(w')$. Thus the edges E_1 are properly embedded inside C . We embed the edges of E_2 outside C as curved line segments. The edges in $A_2 \cap E_2$, if any, can be drawn without intersecting due to the fact that the segments in A_2 are non-crossing. We can argue that the edges of $E_2 \setminus A_2$ can be embedded without crossing as in the previous case. Observe that the edges in E_3 correspond to a vertex $x/ \in R \cup B$, having $\lambda(x)$ and $\rho(x)$ defined, such that $\text{color}(\lambda(x)) \neq \text{color}(\rho(x))$. Also, observe that neither $(\lambda(x), x)$ nor $(x, \rho(x))$ belong to $E_1 \cup E_2$ by the way E_1 and E_2 are defined. Hence, each edge $(\lambda(x), \rho(x)) \in E_3$ can be drawn as the union of the segments $(\lambda(x), x) \in A_1$ and $(x, \rho(x)) \in A_2$. Therefore, G is planar as all the edges in E are drawn without any intersection.

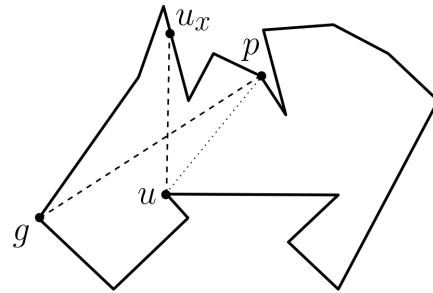
(b) Let x be any vertex of P . We will show that there exists $r \in R$ and $b \in B$, such that r sees x , b sees x , and $(r, b) \in E$. We argue by distinguishing between the following two cases. Case (i) $x \notin R \cup B$: If both $\lambda(x)$ and $\rho(x)$ are defined and $\text{color}(\lambda(x)) \neq \text{color}(\rho(x))$, then $(\lambda(x), \rho(x)) \in E_3$, and hence the claim is true. If both $\lambda(x)$ and $\rho(x)$ are defined and $\text{color}(\lambda(x)) = \text{color}(\rho(x))$, then there must exist $w \in R \cup B$ such that w sees x , and $\text{color}(w)$ is different from $\text{color}(\lambda(x))$ and $\text{color}(\rho(x))$. Without loss of generality we assume that w be the first such vertex while traversing P 's boundary clockwise from u , and $\lambda(x) \prec w \prec x$ (a similar argument works if $x \prec w \prec \rho(x)$ assuming w be the last such vertex while traversing P 's boundary clockwise from u). Let $(\lambda(y), y) \in A_1$ associate with w , implies $\lambda(x) \prec \lambda(y) \prec w \prec y \prec x$. If $y \neq x$, then $\lambda(y)$ sees x due to order claim on the vertices $\lambda(y), w, y$, and x . If $y = x$, then $\lambda(x) = \lambda(y)$, so $\lambda(y)$ sees x . Since w is the first vertex that sees x and satisfies $\lambda(x) \prec w \prec \rho(x)$, it must be that $\text{color}(\lambda(y)) \neq \text{color}(w)$. So the edge $(\lambda(y), w) \in E_1$. Hence, the claim is true. The above argument works even if only $\lambda(x)$ (or $\rho(x)$) is defined.

Case (ii) $x \in R \cup B$: If $\lambda(x)$ is defined and if $\text{color}(\lambda(x)) \neq \text{color}(x)$, then $(\lambda(x), x) \in E_1$, and hence the claim is true. Similarly, if $\rho(x)$ is defined and if $\text{color}(\rho(x)) \neq \text{color}(x)$, then $(x, \rho(x)) \in E_2$, and hence the claim is true. Suppose $\lambda(x)$ is defined, but $\text{color}(\lambda(x)) = \text{color}(x)$ (the argument is similar if $\rho(x)$ is defined and $\text{color}(x) = \text{color}(\rho(x))$). There must exist $w \in R \cup B$ such that w sees x , $\lambda(x) \prec w \prec x$, and $\text{color}(\lambda(x)) \neq \text{color}(w)$. Without loss of generality, let w itself be the first such vertex (while traversing P 's boundary clockwise from u). Now, we can prove the claim by proceeding as in Case (i). \blacktriangleleft

► Lemma 5. *There exists a PTAS for vertex guarding a weakly-visible polygon, with visibility edge $e = (u, v)$ where the angles of u and v are convex, with half-guards where half-guards can only see to the right.*



■ **Figure 5** Removing the convexity assumption.



■ **Figure 6** Any point $p \in P_3$ that is seen by a point $g \in P_2$ is also seen by u .

2.4 Removing the Convexity Assumption

In [3], the convexity assumption is handled by placing guards at u and v . It is proved that every vertex “below” the edge $e = (u, v)$ is seen by either u or v . For half-guards, if guards are similarly placed at u and v the analysis of [3] does not extend to half-guarding. A portion of the polygon below the $e = (u, v)$ edge is unseen, see, for example, P_5 and P_2 from Figure 5. We now show how to remove the convexity assumption of u and v . Let l_u be the vertical line going through u ; u_x and u_y be the points of intersection of l_u with the boundary of P . Let h_v be the horizontal line through v that hits the boundary of P at v_y to the right of v . Let l_v be the vertical line through v that hits the boundary of P at v_x below v . The segments $\overline{u_x u_y}$, $\overline{v v_x}$, and $\overline{v v_y}$ split the polygon into at most five sub-polygons, say P_1, P_2, P_3, P_4, P_5 ; each weakly-visible from some edge (see Fig. 5). Thus, vertex guarding the vertices of each sub-polygon ensures vertex guarding the vertices of P . As shown in [3], if the WV-polygon can be guarded using only a constant number (say c) of vertices, then we try all such possibilities and obtain an optimal solution in $O(n^c)$ time. Assuming the polygon cannot be guarded with a constant number of guards, we place guards at u and v and show that the final guarding set, including these two guards, is a $(1 + \epsilon)$ -approximation.

The entire P_1 region must be seen by the point u on the edge $e = (u, v)$. All the vertices in P_4 are seen by v . Thus, vertex guarding the vertices of P_1 and P_4 can be achieved by placing 1 half-guard at u and v , respectively. Let U be the set of vertices in P_3 that are neither guarded by u nor v . We apply Algorithm 1 on P_3 with the WV-edge \overline{uv} to vertex guard U . Let S_3 be the solution obtained.

Observe that the polygons P_2 and P_5 are weakly-visible from $\overline{u_x u_y}$, and $\overline{v v_x}$, respectively. Hence, Algorithm 1 can be applied individually since u_x, u_y, v , and v_x are convex vertices in their respective sub-polygons. Since Claim 1 (half-guard order claim) holds for these polygons, the A_1 segments in those polygons are non-crossing. Thus, one can construct a graph G for each sub-polygon where G is planar and G satisfies the locality condition. Let S_2 and S_5 be the solutions obtained for P_2 and P_5 , respectively.

It remains to show that the solutions S_2, S_3 and S_5 are disjoint. In other words, the local search algorithm can be run independently on each sub-polygon. No guard in S_2 will see any of the U vertices in P_3 nor P_5 . Similarly, no guard in S_3 will see vertices in P_2 nor P_5 . Lastly, no guard in S_5 will see vertices in P_2 nor the U vertices in P_3 .

It is immediately obvious that no guard in S_3 nor S_5 can see any vertex in P_2 . Any such vertex would need to see left to cross the $\overline{u_x u_y}$ line segment and no guard can see left. In a similar fashion, no guard of S_2 nor S_3 can see any vertex in P_5 . In order to see into P_5 , a

guard must see to the left of the $\overline{vv_x}$ line segment and no guard can see left. When guarding P_3 , we assumed that guards were already placed at u and v and therefore, we only need to run the local search algorithm on the remaining vertex set $U \in P_3$ such that no vertex in U is seen by either u or v .

▷ **Claim 6.** Any vertex $p \in P_3$ that is seen by any boundary point $g \in P_2$ is seen by u .

Proof. By definition, u sees u_x since u_x is directly above u . It must be the case that $u \prec g \preceq u_x$. Since u sees u_x and g sees p , the polygon cannot pierce the $\overline{uu_x}$ nor \overline{gp} line segments. The only way to block u from seeing p is to come from the “other side,” see Fig. 6. However, if blocking occurs in this manner, then p no longer has a line segment connecting it to the \overline{uv} edge, contradicting the fact that the polygon is weakly-visible. Therefore, u must see p . ◁

▷ **Claim 7.** Any vertex $p \in P_3$ that is seen by a boundary point $g \in P_5$ is seen by v .

The set S_2, S_3 or S_5 may not be a feasible solution of P as S_2, S_3 and S_5 may contain guards at u_x, u_y, v_x , or v_y , which are not vertices of P . Assuming v_y is not a vertex, it is not considered by the algorithm for P_3 . The vertex that precedes v_y is considered by the local search algorithm for P_3 . No guard will be placed at v_x when running the local search algorithm on P_5 . A guard at v_x only sees itself and v . A guard was already placed at v making any guard placed at v_x redundant.

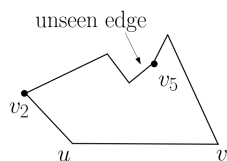
If the local search algorithm places a guard at u_y or u_x when creating a solution for P_2 , then that guard was only placed to guard u_y and/or u_x . Since the rest of the P_2 polygon is to the left of u_x and u_y , guards placed here will not see any other vertex of P_2 . If this happens, then we sweep a vertical line leftward starting at $\overline{u_x u_y}$ until it hits a vertex of P_2 . The guard is moved to this vertex. That vertex will see both u_x and u_y . As shown in Claim 6, neither u_x nor u_y will be responsible for guarding a vertex in P_3 and it is obvious they will not contribute to the other sub-polygons. Therefore, no guards will be placed at u_x, u_y, v_x nor v_y .

► **Theorem 8.** For any weakly-visible polygon, there exists a PTAS for vertex guarding the vertices of the polygon with half-guards.

Proof. Let P be a polygon that is weakly-visible from an edge $e = (u, v)$. We divide the polygon P into at most five sub-polygons as discussed above. Let S_2, S_5 be the set of half-guards obtained by our algorithm for P_2 and P_5 . The obtained set S_3 guards the set of vertices U of P_3 that are not seen by the half-guards placed at u and v . Let OPT be an optimal half-guard set of vertices of P . Let $S = S_2 \cup S_3 \cup S_5 \cup \{u, v\}$.

It can be observed that, the vertices in P_2 (resp. P_5) can only be guarded by half-guards placed at the vertices of P_2 (resp. P_5). Let O_i be an optimal solution for vertex guarding the sub-polygon P_i , and $OPT_i \subset OPT$ be the set of half-guards (in the optimum solution of P) lying in P_i . We get, $|S_i| \leq (1 + \epsilon) \cdot |O_i| \leq (1 + \epsilon) \cdot |OPT_i|$ as $|O_i| \leq |OPT_i|$.

Consider the sub-polygon P_3 . Let O_3 be an optimal solution for vertex guarding the set U in P_3 . Note that, the subset of vertices U is not guarded by $OPT_1 \cup OPT_2 \cup OPT_4 \cup OPT_5$. Let $OPT_3 \subset OPT$ be the set of guards lying in P_3 to guard U . We have, $|O_3| \leq |OPT_3|$, implying $|S_3| \leq (1 + \epsilon) \cdot |O_3| \leq (1 + \epsilon) \cdot |OPT_3|$. Combining the above inequalities, we have $|S| \leq (1 + \epsilon) \cdot |O_2| + (1 + \epsilon) \cdot |O_3| + (1 + \epsilon) \cdot |O_5| + 2$. Thus, $|S| \leq (1 + \epsilon) \cdot OPT + 2$. We can get rid of 2 in the inequality by adjusting k in the algorithm. ◀



■ **Figure 7** The vertices are half-guarded by v_2 and v_5 but an edge remains unseen.

2.5 Guarding the boundary of a weakly-visible polygon

In the previous section, we vertex guarded the vertices of a WV-polygon. However, even though all vertices are seen, it is possible that part of the boundary is unseen (see Fig. 7). Let $AGP(G, W)$ denote the problem of guarding the point set W with minimum number of entries in G . Let ∂P denote the boundary of P . We consider the following problem. Given a weakly-visible polygon P , construct a set $W \subset \partial P$ (witness points) such that an optimal solution for $AGP(V, W)$ is also an optimal solution for $AGP(V, \partial P)$. This discretization is based on Friedrichs et al. [10].

Computation of a witness set

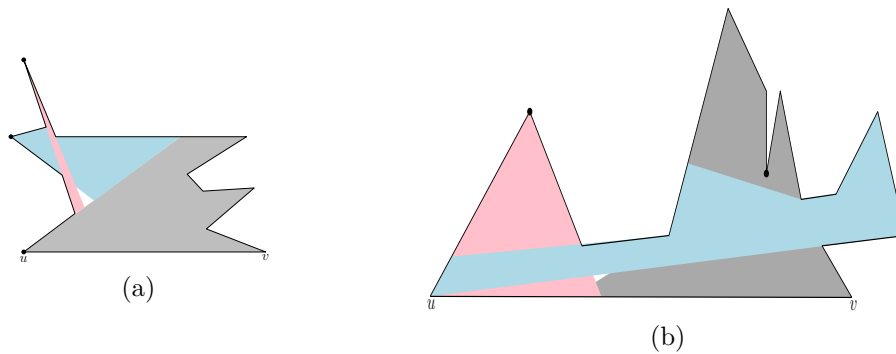
We have a finite guard set V , which are the vertices of P . In order to guard ∂P , we find a discrete set of witness points $W \subset \partial P$. Let $g \in V$ be one of the half-guard candidates and $\mathcal{V}(g)$ be the visibility region of g on P 's boundary ($\mathcal{V}(g) \subseteq \partial P$). $\mathcal{V}(g)$ creates $O(n)$ closed intervals (may be degenerate to a point) along the boundary ∂P . Considering the intervals generated by all the members in the guard set G , we split the entire boundary ∂P into maximal intervals, called *features* such that every point in a feature f is seen by the same set of guards $G(f) = \{g \in V \mid f \subseteq \mathcal{V}(g)\}$. Note that, (i) the union of the features is ∂P , and (ii) if any point in a feature is guarded by some guard in G , then the entire feature is guarded by that half-guard.

We pick any arbitrary point of each feature to the witness set W . So if we guard these $O(n|V|)$ witnesses in W , then the entire feature set, and as a result, ∂P is guarded. We can further reduce the number of witnesses by only using those features f with inclusion-minimal² $G(f)$. Let F be the feature set and G be the guard set (here $G = V$). Let w_f be an arbitrary point in a feature $f \in F$, then the witness of G is defined as $W = \{w_f \mid f \in F, G(f) \text{ is inclusion minimal}\}$.

► **Lemma 9.** *For a weakly-visible polygon P , with vertex set V and guard set G (here $G = V$), any feasible solution of $AGP(G, W)$ is also a feasible solution of $AGP(G, \partial P)$.*

Proof. Let $C \subset G$ be a feasible guard set of the witness set W . Suppose C does not guard some point $p \in \partial P$. Since every point on ∂P is visible to at least one vertex, some half-guard $g \in G$ should guard p . Therefore, p must be part of some feature $f \in F$. The set W either contains some witness $w_f \in f$ or a witness $w_{f'}$ such that $G(f') \subseteq G(f)$ (inclusion-minimality). In the first case, p must be guarded by C , otherwise w_f will not be guarded, and hence C is not a feasible solution for $AGP(G, W)$, leading to a contradiction. In the second case, some half-guard $g \in C$ guards $w_{f'}$. That half-guard should also guard f , thus it guards p ; thus arriving at a contradiction of the statement that p is not guarded. ◀

² There does not exist another feature f' such that $G(f') \subseteq G(f)$



■ **Figure 8** Guards that see the boundary of P but do not see the entire interior of P . (a) Example with half-guards. (b) Example with full-guards.

We apply the algorithm discussed above to guard the witness set W with the guard set V . Thus, we have the following result:

► **Theorem 10.** *For any weakly-visible polygon P , there exists a PTAS for finding the minimum number of half-guards at the vertices of P , that are required to guard the entire boundary of P .*

► **Remark.** Vertex guarding the entire boundary may not necessarily imply vertex guarding the entire polygon. In other words, even if the boundary is fully guarded there may be some pockets (invisible regions) inside the polygon. In Fig. 8(a), any solution (even an optimal one) guarding the boundary must have guards at the tip of the spike for guarding the spikes in the polygon as the guards can see only to the right. If the local search algorithm encounters the guards shown in the figure it will report that guard set. A similar observation can be observed with full-guards too. Note that for the weakly-visible polygon shown in Fig. 8(b), at least three full-guards are necessary to cover the boundary. If the local search algorithm encounters the guards shown in the figure, it will return them and exit as it cannot improve the solution further. In this case, there is a pocket. Indeed, there is another set of three points that cover the entire boundary as well as the interior of the polygon. However, the local search algorithm might not pick them.

3 NP-hardness for Vertex Guarding a Terrain with Half-Guards

Abusing notation, only in this section, we will assume that half-guards can only see “down.” Let $p.y$ be the y -coordinate of a point p . We define seeing *down* as: a point p sees a point q if $p.y \geq q.y$. We begin this section by briefly sketching how the NP-hardness reduction works for terrain guarding with full-guards. The interested reader is encouraged to read [13] to see the full details of the original reduction. The reduction modifications begin in Section 3.1.

The terrain guarding reduction is from PLANAR 3SAT [18] where an instance has n variables and m clauses. The reduction works by assigning vertices on the terrain to truth values of variables from the PLANAR 3SAT instance. For each variable in the PLANAR 3SAT instance, variable gadgets are created such that the gadget contains a vertex representing the literal x_i and a vertex representing the literal \bar{x}_i , see Figure 9(middle). These variable gadgets are grouped together in chunks on the terrain. Figure 9(left) shows an example of one such chunk that contains one variable gadget for each variable in $\{x_1, x_2, \dots, x_n\}$. These chunks are replicated on the terrain such that a guard placed at the vertex representing

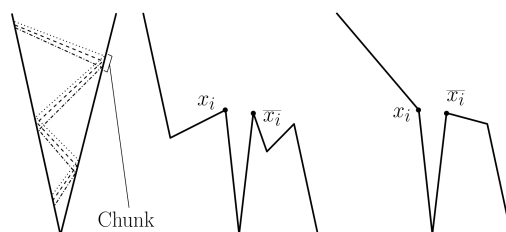
the literal x_i in chunk C_j would require a guard to be placed at the vertex representing the literal x_i in a different chunk C_k . There are points on the terrain that correspond to clauses in the PLANAR 3SAT instance. For example, if clause $c_i = (x_{i+1} \vee \overline{x_{i+3}} \vee x_{i+4})$ were in the original PLANAR 3SAT instance, then a point on the terrain would exist that is seen by three vertices corresponding to the literals x_i , $\overline{x_{i+3}}$ and x_{i+4} . If one of those vertices has a guard placed on it, then the clause would be satisfied. If none of those vertices has a guard placed on it, then an extra guard would be required to guard the terrain. If some minimum number of guards were placed and the entire terrain was seen, then the original instance was satisfied. If not, then the original instance was unsatisfiable.

The terrain guarding reduction only works if guards are full-guards. If the guards are half-guards that only see down, part of the terrain would go unseen. For example, in Figure 9, the portion of the terrain near the top could potentially be unseen. In the original reduction, a guard placed in the highest variable gadget of chunk will see this portion. Other parts of the terrain may also go unseen with the original reduction. Several tweaks and modifications to the reduction need to happen to ensure all of the terrain is guarded.

3.1 Terrain Hardness Modifications

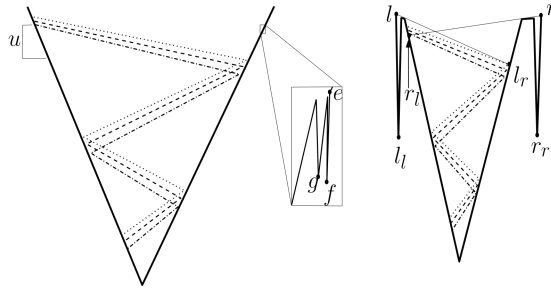
If we restrict guards to be half-guards that see down in the terrain, then the terrain guarding problem with these half-guards is NP-hard. In regular terrain guarding, a point p sees another point q if the line segment connecting p and q does not go below the terrain. In this half-guard variant, the point p sees q only if the y-coordinate of p is greater than or equal to the y-coordinate of q and the line segment connecting p and q does not go below the terrain.

We describe the changes to the gadgets of [13] that must be made in order for the reduction to hold. Each part will explain why the original gadget doesn't work for this half-guard variant and what changes must be made in order to have the reduction hold.



■ **Figure 9** The left shows an overview of the NP-hardness reduction for terrain guarding. The middle is a variable gadget. The right is a starting gadget.

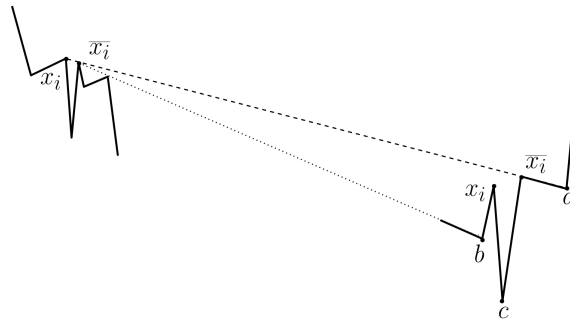
Between Chunks: Let us order the chunks from top to bottom (C_1, C_2, \dots, C_m) . We will assume that all variables gadgets in chunk C_i are below all variable gadgets in chunk C_{i-1} . In the original reduction, the terrain between C_i and C_{i+2} is seen by any guard placed in the C_{i+1} chunk. Since guards can only see down in this variant, a portion of the terrain, which we will call u , below the lowest variable gadget in C_i and above the highest variable gadget in C_{i+1} will be unseen, see Figure 10(left). To fix this, we place a new gadget on the other side of the terrain from C_i . This gadget is at the same y-coordinate of the lowest variable gadget in chunk C_i . This ensures that the new guard will not affect the mirroring. The vertices f and g can be seen from vertex e and by no other point outside of this small region. Placing a guard at e will see f and g and also see the unseen region u below chunk C_i .



■ **Figure 10** (Left) To ensure the entire terrain is seen, a guard is placed at e to see vertices f and g along with the u region. (Right) The overview of the terrain reduction where guards see down.

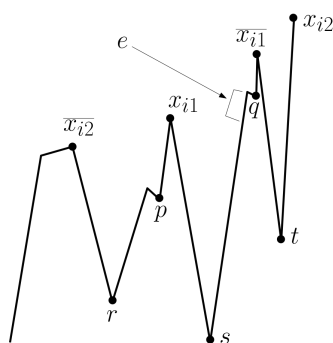
Variable Gadget: The following minor modifications need to be done in the variable gadgets to work for proving the hardness for half-guards.

Assume the \bar{x}_i vertex in the variable gadget in chunk C_j has a guard placed on it and it sees b . In this case, a guard is placed at \bar{x}_i in the variable gadget in chunk C_{j+1} to see a and c . The entire x_i variable gadget in chunk C_{j+1} is seen. Likewise, if a guard placed at x_i in chunk C_j sees a , then a guard is placed at x_i in chunk C_{j+1} sees b and c . A small portion of the terrain below \bar{x}_i in chunk C_{j+1} may have been missed, see Figure 11. However, if the \bar{x}_i vertex in chunk C_j is lowered just slightly, then the guard placed at x_i in C_j will see this region and an additional guard is not required. One must ensure the previously placed x_i does not see b but it can see anything in this gadget above b . Therefore, we ensure that \bar{x}_i in the previous variable gadget is placed in such a way that it blocks x_i from seeing just above b in the subsequent variable gadget.



■ **Figure 11** The variable gadget remains mostly unchanged from [13].

Removing a Variable: We will assume we are removing a variable from chunk C_i . When a variable gadget is removed going upwards, the gadget is modified slightly to remove the a and b vertices. Such a gadget is also called a starting gadget. When a guard is placed at the “lower” vertex in this gadget, a small portion of the terrain below the “higher” vertex remains unseen. In Figure 9(right), a starting gadget is shown. A guard placed at \bar{x}_i would not see the small portion of the terrain below the x_i guard. To ensure this region is seen, we look at the variable patterns placed in the chunk above it, chunk C_{i-1} . The guard placed in the lowest variable pattern will see all of these potentially unseen portions. For example, in Figure 10(right), the guard placed in the variable gadget, x_k , directly above the r_l point will see all of these unseen regions in the variable patterns between l_r and the variable gadget for x_k in chunk C_2 . Therefore, no modification is needed and no additional guard is needed.



■ **Figure 12** The inversion gadget for variable x_i in chunk C_{j+1} . This gadget is not tweaked but the variable gadget x_i in chunk C_j is modified slightly.

When removing a variable going down, the variable gadget is simply removed. The guard placed in the previous chunk's lowest variable gadget will see this region. If the lowest variable was being removed, then the e guard in the gadget that sees between chunks will see this region (see Figure 10(left)).

Above chunk C_1 and C_2 : Since guards cannot see “up,” a guard must be placed that guards the terrain above the first set of variable gadgets in chunk C_1 and above the variable gadgets in C_2 . Assume without loss of generality that C_1 is on the left side of the terrain. Two guards are placed at points l and r as shown in Figure 10(right). These guards are required to see their own set of distinguished points, namely l_l, l_r, r_r and r_l . They see the “top” of the terrain above chunks C_1 and C_2 . Since all gadgets in chunk C_1 are starting gadgets, r_l is placed below chunk C_1 to ensure the relevant part of the terrain in the starting gadgets and the terrain above the chunk are seen.

Inversion Gadgets: The updating required for the inversion gadgets are stated below:

See Figure 12 for a sample inversion gadget placed in chunk C_{j+1} . If a guard from the variable gadget x_i in chunk C_j sees point p , then when guards are placed at $\overline{x_{i1}}$ and $\overline{x_{i2}}$, the entire gadget is seen. If the guard from chunk C_j guard sees point q , then guards are placed at x_{i1} and x_{i2} . This leaves a small portion of the terrain unseen, the line segment e in Figure 12. To fix this, similar to the tweak of the variable gadget, the previously placed guard in chunk C_j is tweaked such that it sees just over the x_{i1} guard to see e . In this example, the previously placed guard must see q and not see p . As long as it is blocked from p , this is all that matters.

Clause Gadgets: No change is needed for the clause gadgets.

Putting it all together: As seen above, certain tweaks and updates are made to ensure that the entire terrain is seen. Making these changes will cause the minimum number of guards that must be placed, k , to increase by $m + 1$. An additional $m - 1$ guards are needed to see the unseen regions between chunks. Two additional guards are added at l and r to see the “top” of the terrain. This gives us a total of $m - 1 + 2 = m + 1$ additional required guards. None of these additional required guards see any of the original distinguished points of the terrain. They see their own set of distinguished points and also see the portion of the terrain that would have been unseen. As shown in [13], if k guards can guard the entire terrain, the instance is satisfiable. If more than k are needed, then the instance is not satisfiable.

► **Theorem 11.** *Finding the smallest vertex guard cover for guarding a terrain using half-guards that see down is NP-hard.*

4 NP-hardness for Vertex Guarding a WV-Polygon with Half-Guards

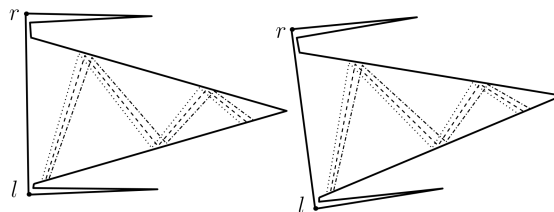
In this section, we give an overview of how to tweak the reduction from the previous section to show that vertex guarding a WV-polygon with half-guards is NP-hard. We will show how to use the terrain guarding hardness result for guards that only see down to show that vertex guarding a WV-polygon with half-guards that see to the right is NP-hard. One can take the modified terrain reduction, rotate it counterclockwise 90° and connect vertex l to vertex r to create a WV-polygon that is visible from the edge $e = (l, r)$, see Figure 13(left). The reduction holds the same way that it does for vertex guarding a terrain with half-guards that only see down.

One will notice that if the WV-edge is rotated slightly counterclockwise, the reduction still holds, see Figure 13(right). The key visibilities from the guards remain and the polygon is still weakly-visible. To help with seeing this, consider the variable patterns of Figure 14. The WV-polygon can be rotated, for example, 10° counterclockwise and the x_i and \bar{x}_i guards still sees “right” to the distinguished points that they needs to see. The reduction begins to fail whenever a guard visibility from the original reduction starts to have a negative slope. In other words, if some point that guard x_i is supposed to see is to the left of x_i . If this happens, the guard no longer sees the distinguished point(s) to its right. To account for this, the original polygon is “stretched” further up-and-to-the-right such that none of the guards visibility have a negative slope. We now give a sketch of how to stretch the terrain/polygon while still maintaining the hardness reduction.

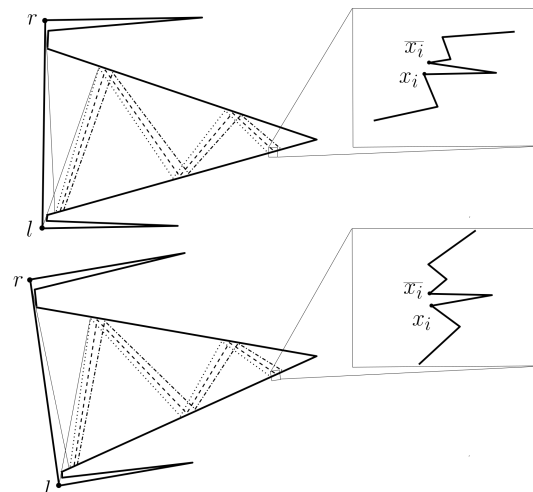
4.1 Stretching Hardness

For the terrain guarding hardness reduction, if half-guards see down and the terrain is rotated 1° in a counterclockwise direction, the terrain does not need to be modified much in order to maintain visibilities. The e guards placed to see between chunks C_i and C_{i+2} need to be shifted “up” to see the correct portion of the terrain between the chunks, see Figure 10. All gadgets are stretched but their visibilities do not change. A similar stretching idea is done for WV-polygons.

An example of stretching a WV-polygon is seen in Figure 15. In the bottom part of this Figure, the WV-edge is parallel to the x-axis and visibility to distinguished points are still to the right. As seen in the example, the original right-most vertex of the original WV-polygon is pulled “up-and-to-the-right” to ensure the polygon remains weakly-visible and all important lines of sight look to the right. The l and r vertices are tweaked slightly to ensure they also see their respective distinguished vertices to the right. Each gadget is also stretched “up-and-to-the-right” to maintain visibilities. An example of such a stretching

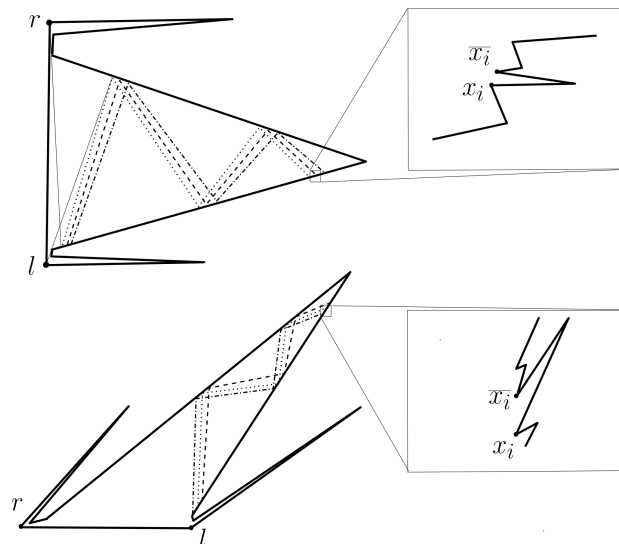


■ **Figure 13** An overview of NP-hardness for WV-polygons.



■ **Figure 14** The top shows an original variable gadget. The bottom shows a rotated variable gadget.

is shown in Figure 15(bottom). In the original reduction, Figure 15(top), the x_i and \bar{x}_i variables see right but they also see “up-and-to-the-right” to mirror its value to the next variable gadget. In the stretched WV-polygon, the variable gadget is stretched but the visibility of x_i and \bar{x}_i that see “up-and-to-the-right” still exist. Those lines of sight are simply steeper than in the original reduction.



■ **Figure 15** A stretched WV-polygon with a rotated variable gadget.

This stretching of the polygon works even if the WV-edge has a positive slope. The polygon can continue to be stretched as far “up-and-right” as necessary. However, the tweak fails when the WV-edge is a vertical edge and the inside of the polygon is to the left of the edge. We leave this as an open problem.

5 Conclusion and Future Work

In this paper, we present a PTAS for vertex guarding the vertices and boundary of a WV-polygon with half-guards that see to the right. This algorithm works regardless of the orientation of the WV-edge. We present an NP-hardness proof for vertex guarding a terrain with half-guards that see down. In contrast, if the half-guards see to the right for vertex guarding a terrain, the problem is polynomial time solvable. We also present an NP-hardness proof for vertex guarding a WV-polygon with half-guards that see to the right. Such a proof works for all instances except when the WV-edge is parallel to the y-axis and the “inside” of the polygon is to the left of the WV-edge. Whether or not this problem is NP-hard is left as an open problem. Future work might include finding a better approximation for the point guarding version of this problem. Insights provided in this paper may help with guarding polygons where the guard can choose to see either left or right, or in other natural directions. One may also be able to use these ideas when allowing guards to see 180° but guards can choose their own direction, i.e. 180° -floodlights.

References

- 1 Alok Aggarwal. *The art gallery theorem: its variations, applications and algorithmic aspects*. PhD thesis, The Johns Hopkins University, 1984.
- 2 Stav Ashur, Omrit Filtser, and Matthew Katz. A constant-factor approximation algorithm for vertex guarding a WV-polygon. *Journal of Computational Geometry*, 12(1), December 2021. doi:10.20382/jocg.v12i1a6.
- 3 Stav Ashur, Omrit Filtser, Matthew J. Katz, and Rachel Saban. Terrain-like graphs: PTASs for guarding weakly-visible polygons and terrains. *Computational Geometry*, 101:101832, 2022. doi:10.1016/j.comgeo.2021.101832.
- 4 Pritam Bhattacharya, Subir Ghosh, and Bodhayan Roy. Approximability of guarding weak visibility polygons. *Discrete Applied Mathematics*, 228, February 2017. doi:10.1016/j.dam.2016.12.015.
- 5 Danny Z. Chen, Vladimir Estivill-Castro, and Jorge Urrutia. Optimal guarding of polygons and monotone chains. In *Proceedings of the 7th Canadian Conference on Computational Geometry, Quebec City, Quebec, Canada, August 1995*, pages 133–138. Carleton University, Ottawa, Canada, 1995. URL: http://www.cccg.ca/proceedings/1995/cccg1995_0022.pdf.
- 6 Marcelo C. Couto, Pedro J. de Rezende, and Cid C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *Int. Trans. Oper. Res.*, 18(4):425–448, 2011. doi:10.1111/j.1475-3995.2011.00804.x.
- 7 Nandhana Duraisamy, Ramesh K. Jallu, Anil Maheshwari, and Subhas C. Nandy. A PTAS for vertex guarding weakly-visible polygons using half-guards. Unpublished Manuscript, 2019.
- 8 Stephan Eidenbenz. Inapproximability results for guarding polygons without holes. In *ISAAC*, pages 427–436, 1998. doi:10.1007/3-540-49381-6_45.
- 9 Khaled M. Elbassioni, Erik Krohn, Domagoj Matijevec, Julián Mestre, and Domagoj Severdija. Improved approximations for guarding 1.5-dimensional terrains. *Algorithmica*, 60(2):451–463, 2011. doi:10.1007/s00453-009-9358-4.
- 10 Stephan Friedrichs, Michael Hemmer, James King, and Christiane Schmidt. The continuous 1.5D terrain guarding problem: Discretization, optimal solutions, and PTAS. *J. Comput. Geom.*, 7(1):256–284, 2016. doi:10.20382/jocg.v7i1a13.
- 11 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997. doi:10.1007/BF02770871.
- 12 Hannah Miller Hillberg, Erik Krohn, and Alex Pahlow. Guarding weakly-visible polygons with half-guards. *Iranian Conference on Computational Geometry*, 2022. URL: <https://iccg2022.ce.sharif.edu/files/Papers/4-Guarding-Weakly-Visible-Polygons-with-Half-Guards.pdf>.

- 13 James King and Erik Krohn. Terrain guarding is NP-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011. doi:10.1137/100791506.
- 14 Erik Krohn. Survey of terrain guarding and art gallery problems. Comprehensive Exam, University of Iowa, November 2007. URL: <http://faculty.cs.uwosh.edu/faculty/krohn/papers/comprehensive.pdf>.
- 15 Erik Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013. doi:10.1007/s00453-012-9653-3.
- 16 Alexander Kröller, Tobias Baumgartner, Sándor P. Fekete, and Christiane Schmidt. Exact solutions and bounds for general art gallery problems. *ACM J. Exp. Algorithmics*, 17, May 2012. doi:10.1145/2133803.2184449.
- 17 D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inform. Theory*, 32(2):276–282, March 1986.
- 18 David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11:329–343, 1982.
- 19 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. QPTAS for geometric set-cover problems via optimal separators. *CoRR*, abs/1403.0835, 2014. arXiv:1403.0835.

A Structural and Algorithmic Study of Stable Matching Lattices of “Nearby” Instances, with Applications

Rohith Reddy Gangam ✉

University of California, Irvine, CA, USA

Tung Mai ✉

Adobe Research, San Jose, CA, USA

Nitya Raju ✉

University of California, Irvine, CA, USA

Vijay V. Vazirani ✉

University of California, Irvine, CA, USA

Abstract

Recently [18] identified and initiated work on a new problem, namely understanding structural relationships between the lattices of solutions of two “nearby” instances of stable matching. They also gave an application of their work to finding a *robust stable matching*. However, the types of changes they allowed in going from instance A to B were very restricted, namely any one agent executes an *upward shift*.

In this paper, we allow any one agent to permute its preference list *arbitrarily*. Let M_A and M_B be the sets of stable matchings of the resulting pair of instances A and B , and let \mathcal{L}_A and \mathcal{L}_B be the corresponding lattices of stable matchings. We prove that the matchings in $M_A \cap M_B$ form a sublattice of both \mathcal{L}_A and \mathcal{L}_B and those in $M_A \setminus M_B$ form a join semi-sublattice. These properties enable us to obtain a polynomial time algorithm for not only finding a stable matching in $M_A \cap M_B$, but also for obtaining the partial order, as promised by Birkhoff’s Representation Theorem [7]. As a result, we can generate all matchings in this sublattice.

Our algorithm also helps solve a version of the robust stable matching problem. We discuss another potential application, namely obtaining new insights into the incentive compatibility properties of the Gale-Shapley Deferred Acceptance Algorithm.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases stable matching, robust solutions, finite distributive lattice, Birkhoff’s Representation Theorem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.19

Related Version *Full Version:* <https://arxiv.org/pdf/1804.05537.pdf>

Funding *Vijay V. Vazirani:* This work was supported in part by NSF grant CCF-1815901.

1 Introduction

The seminal 1962 paper of Gale and Shapley [14] introduced the stable matching problem and gave the Deferred Acceptance (DA) Algorithm for it. In the process, they initiated the field of matching-based market design. Over the years, numerous researchers unearthed the remarkably deep and pristine structural properties of this problem – this led to polynomial time algorithms for numerous problems, in particular those addressing various operations related to the lattice of stable matchings, see details below as well as in the books [17, 15, 20, 22, 12].



© Rohith Reddy Gangam, Tung Mai, Nitya Raju, and Vijay V. Vazirani;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 19; pp. 19:1–19:20



Leibniz International Proceedings in Informatics

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

Recently [18] identified and initiated work on a new problem which appears to be fundamental and deserving of an in-depth study, namely understanding structural relationships between the lattices of solutions of two “nearby” instances. [18] had given an application of their work to finding a *robust stable matching* as described below. Let us say that two instance A and B of stable matching are *nearby instances* if B is obtained from A when one agent changes their preference list. Such pairs of instances arise naturally in an even more important context: the study of incentive compatibility of the DA Algorithm: one of the agents manipulates its preference list in order to get a better match. The types of manipulations allowed in [18] were very restricted, namely any one agent executes an *upward shift*, see definition below. They left the open problem of tackling more general changes.

[21] showed that finding a stable matching across k (≥ 2) arbitrary instances is NP-Hard. In this paper, we allow any one agent to permute its preference list *arbitrarily*. Let A and B be the resulting pair of instances, let M_A and M_B be the sets of their stable matchings and \mathcal{L}_A and \mathcal{L}_B be the corresponding lattices of stable matchings. We prove that the matchings in $M_A \cap M_B$ form a sublattice of both \mathcal{L}_A and \mathcal{L}_B and those in $M_A \setminus M_B$ form a join semi-sublattice, see definitions in Section 1.1. This enables us to obtain a polynomial time algorithm for not only finding a stable matching in $M_A \cap M_B$, but also to obtain the partial order, promised by Birkhoff’s Representation Theorem [7], which helps generate all matchings in this sublattice. We also apply our algorithm to a more general setting for robust stable matching than the one given in [18].

The setting defined in [18] was the following: Let A be an instance of stable matching on n workers and n firms. A *domain of errors*, D , is defined via an operation called *upward shift*: For a firm f , assume its preference list in instance A is $\{\dots, w_1, w_2, \dots, w_k, w, \dots\}$. Move up the position of worker w so f ’s list becomes $\{\dots, w, w_1, w_2, \dots, w_k, \dots\}$. An analogous operation is defined on a worker w ’s list; again some firm f on its list is moved up. For each firm and each worker, consider all possible shifts to get the domain D ; clearly, $|D| = \binom{2n}{1} \binom{n}{2} = O(n^3)$. Assume that *one error* is chosen from D via a given discrete probability distribution over D to obtain instance B . A *robust stable matching* is a matching that is stable for A and maximizes the probability of being stable for B . A polynomial time algorithm was given for finding such a matching.

Since we allow an *arbitrary permutation* to be applied to any *one worker or any one firm’s* preference list, our domain of errors, say T , has size $2n(n!)$. Let $S \subseteq T$ and define a *fully robust stable matching w.r.t. S* to be a matching that is stable for A and for *each* of the $|S|$ instances obtained by introducing one error from S . We give an $O(|S|p(n))$ algorithm to determine if such a matching exists and if so to find one, where p is a polynomial function. In particular, if S is polynomial sized, then our algorithm runs in polynomial time. Clearly, this notion is weaker than the previous one, since we cannot extend it to the probabilistic setting; we leave that as an open problem, see Section 8.

In case all errors in S are on one side only, say the firms, it turns out that Algorithm D, which is a simple modification of the Deferred Acceptance Algorithm, works; this algorithm is given in Appendix D. However, extending this algorithm to the case that errors occur on both sides, workers and firms, results in an algorithm (Algorithm D) that has exponential runtime. Our polynomial time algorithm follows from a study of the sublattices of the lattice of stable matchings.

Conway, see [17], proved that the set of stable matchings of an instance forms a finite distributive lattice; see definitions in Section 2.2. Knuth [17] asked if every finite distributive lattice is isomorphic to the lattice arising from an instance of stable matching. A positive answer was provided by Blair [8]; for a much better proof, see [15]. A key fact about such

lattices is Birkhoff's Representation Theorem [7], which has also been called *the fundamental theorem for finite distributive lattices*, e.g., see [23]. It states that corresponding to such a lattice, \mathcal{L} , there is a partial order, say Π , such that \mathcal{L} is isomorphic to $L(\Pi)$, the lattice of closed sets of Π (see Section 2.2 for details). We will say that Π *generates* \mathcal{L} .

The following important question arose in the design of our algorithm: For a specified sublattice \mathcal{L}' of \mathcal{L} , obtain partial order Π' from Π such that Π' generates \mathcal{L}' . Our answer to this question requires a study of Birkhoff's Theorem from this angle; we are not aware of any previous application of Birkhoff's Theorem in this manner. We define a set of operations called compressions; when a compression is applied to a partial order Π , it yields a partial order Π' on (weakly) fewer elements. The following implication of Birkhoff's Theorem is useful for our purposes:

► **Theorem 1.** *There is a one-to-one correspondence between the compressions of Π and the sublattices of $L(\Pi)$ such that if sublattice \mathcal{L}' of $L(\Pi)$ corresponds to compression Π' , then \mathcal{L}' is generated by Π' .*

A proof of Theorem 1, using stable matching lattices, is given in Section B for completeness. In the case of stable matchings, Π can be defined using the notion of *rotations*; see Section 2.2 for a formal definition. Since the total number of rotations of a stable matching instance is at most $O(n^2)$, Π has a succinct description even though \mathcal{L} may be exponentially large. Our main algorithmic result is:

► **Theorem 2.** *There is an algorithm for checking if there is a fully robust stable matching w.r.t. any set $S \subseteq T$ in time $O(|S|p(n))$, where p is a polynomial function. Moreover, if the answer is yes, the set of all such matchings forms a sublattice of \mathcal{L} and our algorithm finds a partial order that generates it.*

The importance of the stable matching problem lies not only in its efficient computability but also its good incentive compatibility properties. In particular, Dubins and Freedman [11] proved that the DA Algorithm is *dominant-strategy incentive compatible (DSIC)* for the proposing side. This opened up the use of this algorithm in a host of highly consequential applications, e.g., matching students to public schools in big cities, such as NYC and Boston, see [3, 1, 2]. In this application, the proposing side is taken to be the students; clearly, their best strategy is to report preference lists truthfully and not waste time and effort on “gaming” the system. In Section 8 we give a hypothetical situation regarding incentive compatibility in which Theorem 2 plays a role.

1.1 Overview of structural and algorithmic ideas

We start by giving a short overview of the structural facts proven in [18]. Let A and B be two instances of stable matching over n workers and n firms, with sets of stable matchings \mathcal{M}_A and \mathcal{M}_B , and lattices \mathcal{L}_A and \mathcal{L}_B , respectively. Let Π be the poset on rotations such that $L(\Pi) = \mathcal{L}_A$; in particular, for a closed set S , let $M(S)$ denote the stable matching corresponding to S . It is easy to see that if B is obtained from A by changing (upshifts only) the lists of only one side, either workers or firms, but not both, then the matchings in $\mathcal{M}_A \cap \mathcal{M}_B$ form a sublattice of each of the two lattices (Proposition 6). Furthermore, if B is obtained by applying a shift operation, then $\mathcal{M}_{A \setminus B} = \mathcal{M}_A \setminus \mathcal{M}_B$ is also a sublattice of \mathcal{L}_A . Additionally, there is at most one rotation, ρ_{in} , that leads from $\mathcal{M}_A \cap \mathcal{M}_B$ to $\mathcal{M}_{A \setminus B}$ and at most one rotation, ρ_{out} , that leads from $\mathcal{M}_{A \setminus B}$ to $\mathcal{M}_A \cap \mathcal{M}_B$; moreover, these rotations can be found in polynomial time. Finally, for a closed set S of Π , $M(S)$ is stable for instance B iff $\rho_{\text{in}} \in S \Rightarrow \rho_{\text{out}} \in S$.

With a view to extending the results of [18], we consider the following abstract question. Suppose instance B is such that $\mathcal{M}_A \cap \mathcal{M}_B$ and $\mathcal{M}_{A \setminus B}$ are both sublattices of \mathcal{L}_A , i.e., \mathcal{M}_A is partitioned into two sublattices. Then, is there a polynomial time algorithm for finding a matching in $\mathcal{M}_A \cap \mathcal{M}_B$? Our answer to this question is built on the following structural fact: There exists a sequence of rotations $r_0, r_1, \dots, r_{2k}, r_{2k+1}$ such that a closed set of Π generates a matching in $\mathcal{M}_A \cap \mathcal{M}_B$ iff it contains r_{2i} but not r_{2i+1} for some $0 \leq i \leq k$ (Proposition 19). Furthermore, this sequence of rotations can be found in polynomial time (see Section 4). Our generalization of Birkhoff’s Theorem described in the Introduction is an important ingredient in this algorithm. At this point, we do not know of any concrete error pattern, beyond shift, for which this abstract setting applies.

Next, we address the case that $\mathcal{M}_{A \setminus B}$ is not a sublattice of \mathcal{L}_A . We start by proving that if B is obtained by permuting the preference list of any one worker, then $\mathcal{M}_{A \setminus B}$ must be a join semi-sublattice of \mathcal{L}_A (Lemma 31); an analogous statement holds if the preference list of any one firm is permuted. Hence we study a second abstract question, namely lattice \mathcal{L}_A is partitioned into a sublattice and a join semi-sublattice (see Section 5). These two abstract questions are called **Setting I** and **Setting II**, respectively, in this paper.

For Setting II, we characterize a compression that yields a partial order Π' , such that Π' generates the sublattice consisting of matchings in $\mathcal{M}_A \cap \mathcal{M}_B$ (Theorem 20). We also characterize closed sets of Π such that the corresponding matchings lie in this sublattice; however, the characterization is too elaborate to summarize succinctly (see Proposition 25). Edges forming the required compression can be found in polynomial time (Theorem 29), hence leading to an efficient algorithm for finding a matching in $\mathcal{M}_A \cap \mathcal{M}_B$.

Finally, consider the setting given in the Introduction, with T being the super-exponential set of all possible errors that can be introduced in instance A and $S \subset T$. We show that the set of all matchings that are stable for A and for each of the instances obtained by introducing one error from S forms a sublattice of \mathcal{L} and we obtain a compression of Π that generates this sublattice (Section 7.2). Each matching in this sublattice is a fully robust stable matching. Furthermore, given a weight function on all worker-firm pairs, we can obtain, using the algorithm of [19], a maximum (or minimum) weight fully robust stable matching.

2 Preliminaries

2.1 The stable matching problem and the lattice of stable matchings

The stable matching problem takes as input a set of workers $\mathcal{W} = \{w_1, w_2, \dots, w_n\}$ and a set of firms $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$; each agent has a complete preference ranking over the set of opposite side. A matching M is a one-to-one correspondence between \mathcal{W} and \mathcal{F} . For each pair $wf \in M$, w is called the partner of f in M (or M -partner) and vice versa. For a matching M , a pair $wf \notin M$ is said to be *blocking* if they prefer each other to their partners. A matching M is *stable* if there is no blocking pair for M .

Let M and M' be two stable matchings. We say that M *dominates* M' , denoted by $M \preceq M'$, if every worker weakly prefers his partner in M to M' . Define the relation *predecessor* as the transitive closure of dominates. The set of stable matchings forms a finite distributive lattice under the above definition of predecessor. The lattice contains a matching, M_0 , that dominates all others and a matching M_z that is dominated by all others. M_0 is called the *worker-optimal matching*, since in it, each worker is matched to his most favorite firm among all stable matchings. Similarly, M_z is *firm-optimal matching*.

2.2 Birkhoff's Theorem and rotations

It is easy to see that the family of closed sets (also called lower sets, Definition 5) of a partial order, say Π , is closed under union and intersection and forms a distributive lattice, with join and meet being these two operations, respectively; let us denote it by $L(\Pi)$. Birkhoff's theorem [7], states that corresponding to any finite distributed lattice, \mathcal{L} , there is a partial order, say Π , whose lattice of closed sets $L(\Pi)$ is isomorphic to \mathcal{L} , i.e., $\mathcal{L} \cong L(\Pi)$. We will say that Π *generates* \mathcal{L} .

One way to define the partial orders generating stable matching lattices is using the concept of *rotation*. For a worker w let $s_M(w)$ denote the first firm f on w 's list such that f strictly prefers w to her M -partner. Let $next_M(w)$ denote the partner in M of firm $s_M(w)$. A *rotation* ρ *exposed* in M is an ordered list of pairs $\{w_0f_0, w_1f_1, \dots, w_{r-1}f_{r-1}\}$ such that for each i , $0 \leq i \leq r-1$, w_{i+1} is $next_M(w_i)$, where $i+1$ is taken modulo r . M/ρ is defined to be a matching in which each worker not in a pair of ρ stays matched to the same firm and each worker w_i in ρ is matched to $f_{i+1} = s_M(w_i)$. It can be proven that M/ρ is also a stable matching. The transformation from M to M/ρ is called the *elimination* of ρ from M .

Let $\rho = \{w_0f_0, w_1f_1, \dots, w_{r-1}f_{r-1}\}$ be a rotation. For $0 \leq i \leq r-1$, we say that ρ *moves* w_i *from* f_i *to* f_{i+1} , and *moves* f_i *from* w_i *to* w_{i-1} . If f is either f_i or is strictly between f_i and f_{i+1} in w_i 's list, then we say that ρ *moves* w_i *below* f . Similarly, ρ *moves* f_i *above* w if w is w_i or between w_i and w_{i-1} in f_i 's list.

2.3 The rotation poset

A rotation ρ' is said to *precede* another rotation ρ , denoted by $\rho' \prec \rho$, if ρ' is eliminated in every sequence of eliminations from M_0 to a stable matching in which ρ is exposed. Thus, the set of rotations forms a partial order via this precedence relationship. The partial order on rotations is called *rotation poset* and denoted by Π .

► **Lemma 3** ([15], Lemma 3.2.1). *For any worker w and firm f , there is at most one rotation that moves w to f , w below f , or f above w . Moreover, if ρ_1 moves w to f and ρ_2 moves w from f then $\rho_1 \prec \rho_2$.*

► **Lemma 4** ([15], Lemma 3.3.2). *Π contains at most $O(n^2)$ rotations and can be computed in polynomial time.*

► **Definition 5.** *A closed set of a poset is a set S of elements of the poset such that if an element is in S then all of its predecessors are also in S .*

There is a one-to-one relationship between the stable matchings and the closed subsets of Π . Given a closed set S , the corresponding matching M is found by eliminating the rotations starting from M_0 according to the topological ordering of the elements in the set S . We say that S *generates* M .

Let S be a subset of the elements of a poset, and let v be an element in S . We say that v is a *minimal* element in S if there are no predecessors of v in S . Similarly, v is a *maximal* element in S if it has no successors in S . The *Hasse diagram* of a poset is a directed graph with a vertex for each element in the poset, and an edge from x to y if $x \prec y$ and there is no z such that $x \prec z \prec y$. In other words, all precedences implied by transitivity are suppressed.

2.4 Sublattice and semi-sublattice

A *sublattice* \mathcal{L}' of a distributive lattice \mathcal{L} is subset of \mathcal{L} such that for any two elements $x, y \in \mathcal{L}$, $x \vee y \in \mathcal{L}'$ and $x \wedge y \in \mathcal{L}'$ whenever $x, y \in \mathcal{L}'$, where \vee and \wedge are the join and meet operations of lattice \mathcal{L} . A *join semi-sublattice* \mathcal{L}' of a distributive lattice \mathcal{L} is subset of \mathcal{L} such that for any two elements $x, y \in \mathcal{L}$, $x \vee y \in \mathcal{L}'$ whenever $x, y \in \mathcal{L}'$. Similarly, *meet semi-sublattice* \mathcal{L}' of a distributive lattice \mathcal{L} is subset of \mathcal{L} such that for any two elements $x, y \in \mathcal{L}$, $x \wedge y \in \mathcal{L}'$ whenever $x, y \in \mathcal{L}'$. Note that \mathcal{L}' is a sublattice of \mathcal{L} iff \mathcal{L}' is both join and meet semi-sublattice of \mathcal{L} .

► **Proposition 6.** *Let A be an instance of stable matching and let B be another instance obtained from A by changing the lists of only one side, either workers or firms, but not both. Then the matchings in $\mathcal{M}_A \cap \mathcal{M}_B$ form a sublattice in each of the two lattices.*

► **Corollary 7.** *Let A be an instance of stable matching and let B_1, \dots, B_k be other instances obtained from A each by changing the lists of only one side, either workers or firms, but not both. Then the matchings in $\mathcal{M}_A \cap \mathcal{M}_{B_1} \cap \dots \cap \mathcal{M}_{B_k}$ form a sublattice in \mathcal{M}_A .*

This corollary gives another justification for Algorithm D, motivated by [21]. This modified Deferred Algorithm works when errors are only on one side. Algorithm D extends this to errors on both sides however it has exponential runtime.

This motivates us to characterize sublattices in the lattice of stable matchings. In Section 7.1, we show that for any instance B obtained by permuting the preference list of one worker or one firm, $\mathcal{M}_{A \setminus B}$ forms a semi-sublattice of \mathcal{L}_A (Lemma 31). In particular, if the list of a worker is permuted, $\mathcal{M}_{A \setminus B}$ forms a join semi-sublattice of \mathcal{L}_A , and if the list of a firm is permuted, $\mathcal{M}_{A \setminus B}$ forms a meet semi-sublattice of \mathcal{L}_A . In both cases, $\mathcal{M}_A \cap \mathcal{M}_B$ is a sublattice of \mathcal{L}_A and of \mathcal{L}_B as shown in Proposition 6.

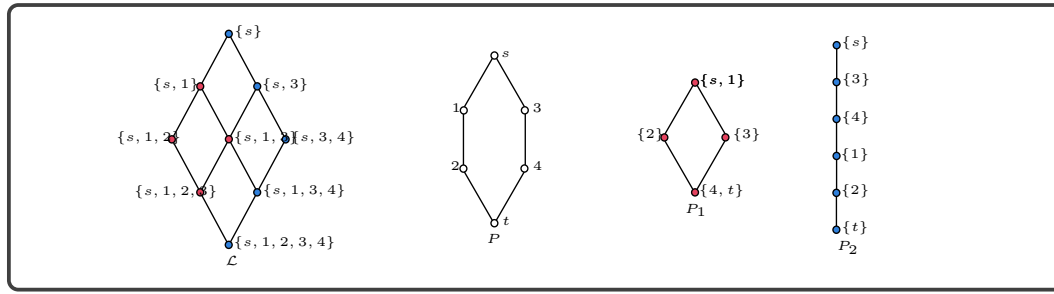
3 Birkhoff’s Theorem on Sublattices

Let Π be a finite poset. For simplicity of notation, in this paper we will assume that Π must have *two dummy elements* s and t ; the remaining elements will be called *proper elements* and the term *element* will refer to proper as well as dummy elements. The element s precedes all other elements and t succeeds all other elements in Π . A *proper closed set* of Π is any closed set that contains s and does not contain t . It is easy to see that the set of all proper closed sets of Π form a distributive lattice under the operations of set intersection and union. We will denote this lattice by $L(\Pi)$. The following has also been called *the fundamental theorem for finite distributive lattices*.

► **Theorem 8** (Birkhoff [7]). *Every finite distributive lattice \mathcal{L} is isomorphic to $L(\Pi)$, for some finite poset Π .*

Our application of Birkhoff’s Theorem deals with the sublattices of a finite distributive lattice. First, in Definition 9 we state the critical operation of *compression of a poset*.

► **Definition 9.** *Given a finite poset Π , first partition its elements; each subset will be called a meta-element. Define the following precedence relations among the meta-elements: if x, y are elements of Π such that x is in meta-element X , y is in meta-element Y and x precedes y , then X precedes Y . Assume that these precedence relations yield a partial order, say Q , on the meta-elements (if not, this particular partition is not useful for our purpose). Let*



■ **Figure 1** Two examples of compressions. Lattice $\mathcal{L} = L(P)$. P_1 and P_2 are compressions of P , and they generate the sublattices in \mathcal{L} , of red and blue elements, respectively. The black edges are directed from top to bottom so higher elements are predecessors of lower elements.

Π' be any partial order on the meta-elements such that the precedence relations of Q are a subset of the precedence relations of Π' . Then Π' will be called a compression of Π . Let A_s and A_t denote the meta-elements of Π' containing s and t , respectively.

For examples of compressions see Figure 1. Clearly, A_s precedes all other meta-elements in Π' and A_t succeeds all other meta-elements in Π' . Once again, by a *proper closed set* of Π' we mean a closed set of Π' that contains A_s and does not contain A_t . Then the lattice formed by the set of all proper closed sets of Π' will be denoted by $L(\Pi')$.

3.1 An alternative view of compression

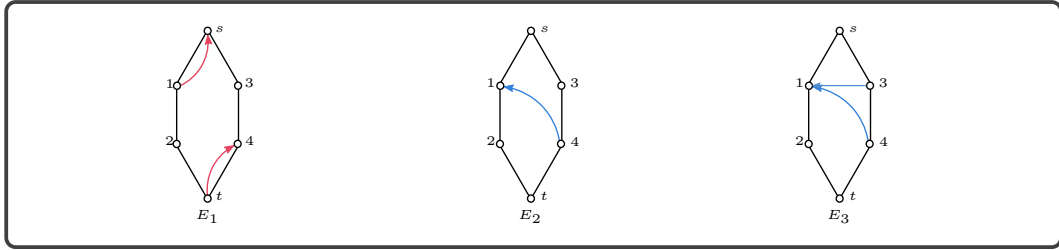
In this section we give an alternative definition of compression of a poset; this will be used in the rest of the paper. The advantage of this definition is that it is much easier to work with for the applications presented later. Its drawback is that several different sets of edges may yield the same compression. Therefore, this definition is not suitable for stating a one-to-one correspondence between sublattices of \mathcal{L} and compressions of Π . Finally we show that any compression Π' obtained using the first definition can also be obtained via the second definition and vice versa (Proposition 10), hence showing that the two definitions are equivalent for our purposes. See Appendix C for more details.

We are given a poset Π for a stable matching instance; let \mathcal{L} be the lattice it generates. Let $H(\Pi)$ denote the Hasse diagram of Π . Consider the following operations to derive a new poset Π' : Choose a set E of directed edges to add to $H(\Pi)$ and let H_E be the resulting graph. Let H' be the graph obtained by shrinking the strongly connected components of H_E ; each strongly connected component will be called a meta-rotation of Π' as defined in Definition 9. The edges which are not shrunk will define a DAG, H' , on the strongly connected components. These edges give precedence relations among meta-rotation for poset Π' .

Let \mathcal{L}' be the sublattice of \mathcal{L} generated by Π' . We will say that the set of edges E defines \mathcal{L}' . It can be seen that each set E uniquely defines a sublattice $L(\Pi')$; however, there may be multiple sets that define the same sublattice. See Figure 2 for examples of sets of edges which define sublattices.

► **Proposition 10.** *The two definitions of compression of a poset are equivalent.*

For a (directed) edge $e = uv \in E$, u is called the *tail* and v is called the *head* of e . Let I be a closed set of Π . Then we say that: I separates an edge $uv \in E$ if $v \in I$ and $u \notin I$; I crosses an edge $uv \in E$ if $u \in I$ and $v \notin I$. If I does not separate or cross any edge $uv \in E$, I is called a *splitting set* w.r.t. E .



■ **Figure 2** E_1 (red edges) and E_2 (blue edges) define the sublattices in Figure 1, of red and blue elements, respectively. E_2 and E_3 define the same compression and represent the same sublattice. All black edges in E_1, E_2 and E_3 are directed from top to bottom (not shown in the figure).

► **Lemma 11.** *Let \mathcal{L}' be a sublattice of \mathcal{L} and E be a set of edges defining \mathcal{L}' . A matching M is in \mathcal{L}' iff the closed subset I generating M does not separate any edge $uv \in E$.*

► **Remark 12.** We may assume w.l.o.g. that the set E defining \mathcal{L}' is *minimal* in the following sense: There is no edge $uv \in E$ such that uv is not separated by any closed set of Π . Observe that if there is such an edge, then $E \setminus \{uv\}$ defines the same sublattice \mathcal{L}' . Similarly, there is no edge $uv \in E$ such that each closed set separating uv also separates another edge in E .

► **Definition 13.** *W.r.t. an element v in a poset Π , we define four useful subsets of Π : $I_v = \{r \in \Pi : r \prec v\}$, $J_v = \{r \in \Pi : r \preceq v\} = I_v \cup \{v\}$, $I'_v = \{r \in \Pi : r \succ v\}$, $J'_v = \{r \in \Pi : r \succeq v\} = I'_v \cup \{v\}$. Notice that $I_v, J_v, \Pi \setminus I'_v, \Pi \setminus J'_v$ are all closed sets.*

► **Lemma 14.** *Both J_v and $\Pi \setminus J'_u$ separate uv for each $uv \in E$.*

Proof. Since uv is in E , u cannot be in J_v ; otherwise, there is no closed subset separating uv , contradicting Remark 12. Hence, J_v separates uv for all uv in E . Similarly, since uv is in E , v cannot be in J'_u . Therefore, $\Pi \setminus J'_u$ contains v but not u , and thus separates uv . ◀

4 Setting I

Under Setting I, the given lattice \mathcal{L} has sublattices \mathcal{L}_1 and \mathcal{L}_2 that partition \mathcal{L} . The main structural fact for this setting is:

► **Theorem 15.** *Let \mathcal{L}_1 and \mathcal{L}_2 be sublattices of \mathcal{L} such that \mathcal{L}_1 and \mathcal{L}_2 partition \mathcal{L} . Then there exist sets of edges E_1 and E_2 defining \mathcal{L}_1 and \mathcal{L}_2 such that they form an alternating path from t to s .*

We will prove this theorem in the context of stable matchings. Let E_1 and E_2 be any two sets of edges defining \mathcal{L}_1 and \mathcal{L}_2 , respectively. We will show that E_1 and E_2 can be adjusted so that they form an alternating path from t to s , without changing the corresponding compressions.

► **Lemma 16.** *There must exist a path from t to s composed of edges in E_1 and E_2 .*

Let Q be a path from t to s according to Lemma 16. Partition Q into subpaths Q_1, \dots, Q_k such that each Q_i consists of edges in either E_1 or E_2 and $E(Q_i) \cap E(Q_{i+1}) = \emptyset$ for all $1 \leq i \leq k - 1$. Let r_i be the rotation at the end of Q_i except for $i = 0$ where $r_0 = t$.



■ **Figure 3** Examples of: (a) canonical path, and (b) bouquet.

Specifically, $t = r_0 \rightarrow r_1 \rightarrow \dots \rightarrow r_k = s$ in Q . Lemma 11 can be used to show that each Q_i can be replaced by a direct edge from r_{i-1} to r_i , and furthermore, all edges not in Q can be removed.

► **Lemma 17.** *Let Q_i consist of edges in E_α ($\alpha = 1$ or 2). Q_i can be replaced by an edge from r_{i-1} to r_i where $r_{i-1}r_i \in E_\alpha$.*

► **Lemma 18.** *Edges in $E_1 \cup E_2$ but not in Q can be removed.*

By Lemma 17 and Lemma 18, $r_0r_1, \dots, r_{k-2}r_{k-1}, r_{k-1}r_k$ are all edges in E_1 and E_2 and they alternate between E_1 and E_2 . Therefore, we have Theorem 15. An illustration of such a path is given in Figure 3(a).

► **Proposition 19.** *There exists a sequence of rotations $r_0, r_1, \dots, r_{2k}, r_{2k+1}$ such that a closed subset generates a matching in \mathcal{L}_1 iff it contains r_{2i} but not r_{2i+1} for some $0 \leq i \leq k$.*

5 Setting II

Under Setting II, the given lattice \mathcal{L} can be partitioned into a sublattice \mathcal{L}_1 and a semi-sublattice \mathcal{L}_2 . We assume that \mathcal{L}_2 is a join semi-sublattice. Clearly by reversing the order of \mathcal{L} , the case of meet semi-sublattice is also covered. The next theorem, which generalizes Theorem 15, gives a sufficient characterization of a set of edges E defining \mathcal{L}_1 .

► **Theorem 20.** *There exists a set of edges E defining sublattice \mathcal{L}_1 such that:*

1. *The set of tails T_E of edges in E forms a chain in Π .*
2. *There is no path of length two consisting of edges in E .*
3. *For each $r \in T_E$, let $F_r = \{v \in \Pi : rv \in E\}$. Then any two rotations in F_r are incomparable.*
4. *For any $r_i, r_j \in T_E$ where $r_i \prec r_j$, there exists a splitting set containing all rotations in $F_{r_i} \cup \{r_i\}$ and no rotations in $F_{r_j} \cup \{r_j\}$.*

A set E satisfying Theorem 20 will be called a *bouquet*. For each $r \in T_E$, let $L_r = \{rv \mid v \in F_r\}$. Then L_r will be called a *flower*. Observe that the bouquet E is partitioned into flowers. These notions are illustrated in Figure 3(b). The black path, directed from s to t , is the chain mentioned in Theorem 20 and the red edges constitute E . Observe that the tails of edges E lie on the chain. For each such tail, the edges of E outgoing from it constitute a flower.

19:10 A Study of Stable Matching Lattices of “Nearby” Instance

Let E be an arbitrary set of edges defining \mathcal{L}_1 . We will show that E can be modified so that the conditions in Theorem 20 are satisfied. Let S be a splitting set of Π . In other words, S is a closed subset such that for all $uv \in E$, either u, v are both in S or u, v are both in $\Pi \setminus S$. We can now replace paths with single edges as explained below.

► **Lemma 21.** *There is a unique maximal rotation in $T_E \cap S$.*

Denote by r the unique maximal rotation in $T_E \cap S$. Let $R_r = \{v \in \Pi : \text{there is a path from } r \text{ to } v \text{ using edges in } E\}$, $E_r = \{uv \in E : u, v \in R_r\}$, $G_r = \{R_r, E_r\}$. Note that $r \in R_r$. For each $v \in R_r$ there exists a path from r to v and $r \in S$. Since S does not cross any edge in the path, v must also be in S . Therefore, $R_r \subseteq S$.

► **Lemma 22.** *Let $u \in (T_E \cap S) \setminus R_r$ such that $u \succ x$ for $x \in R_r$. Then we can replace each $uv \in E$ with rv .*

Keep replacing edges according to Lemma 22 until there is no $u \in (T_E \cap S) \setminus R_r$ such that $u \succ x$ for some $x \in R_r$.

► **Lemma 23.** *Let $X = \{v \in S : v \succeq x \text{ for some } x \in R_r\}$. Then: $S \setminus X$ is a closed subset; $S \setminus X$ contains u for each $u \in (T_E \cap S) \setminus R_r$; $(S \setminus X) \cap R_r = \emptyset$; $S \setminus X$ is a splitting set.*

► **Lemma 24.** *E_r can be replaced by the following set of edges: $E'_r = \{rv : v \in R_r\}$.*

Proof of Theorem 20. To begin, let $S_1 = \Pi$ and let r_1 be the unique maximal rotation according to Lemma 21. Then we can replace edges according to Lemma 22 and Lemma 24. After replacing, r_1 is the only tail vertex in G_{r_1} . By Lemma 23, there exists a set X such that $S_1 \setminus X$ does not contain any vertex in R_{r_1} and contains all other tail vertices in T_E except r_1 . Moreover, $S_1 \setminus X$ is a splitting set. Hence, we can set $S_2 = S_1 \setminus X$ and repeat.

Let r_1, \dots, r_k be the rotations found in the above process. Since r_i is the unique maximal rotation in $T_E \cap S_i$ for all $1 \leq i \leq k$ and $S_1 \supset S_2 \supset \dots \supset S_k$, we have $r_1 \succ r_2 \succ \dots \succ r_k$. By Lemma 24, for each $1 \leq i \leq k$, E_{r_i} consists of edges $r_i v$ for $v \in R_{r_i}$. Therefore, there is no path of length two composed of edges in E and condition 2 is satisfied. Moreover, r_1, \dots, r_k are exactly the tail vertices in T_E , which gives condition 1.

Let r be a rotation in T_E and consider $u, v \in F_r$. Moreover, assume that $u \prec v$. A closed subset I separating rv contains v but not r . Since I is a closed subset and $u \prec v$, I contains u . Therefore, I also separates ru , contradicting the assumption in Remark 12. The same argument applies when $v \prec u$. Therefore, u and v are incomparable as stated in condition 3.

Finally, let $r_i, r_j \in T_E$ where $r_i \prec r_j$. By the construction given above, $S_j \supset S_{j-1} \supset \dots \supset S_i$, $R_{r_j} \subseteq S_j \setminus S_{j-1}$ and $R_{r_i} \subseteq S_i$. Therefore, S_i contains all rotations in R_{r_i} but none of the rotations in R_{r_j} , giving condition 4 which can be restated as Proposition 25. ◀

► **Proposition 25.** *There exists a sequence of rotations $r_1 \prec \dots \prec r_k$ and a set F_{r_i} for each $1 \leq i \leq k$ such that a closed subset generates a matching in \mathcal{L}_1 if and only if whenever it contains a rotation in F_{r_i} , it must also contain r_i .*

6 Algorithm for Finding a Bouquet

In this section, we give an algorithm for finding a bouquet. Let \mathcal{L} be a distributive lattice that can be partitioned into a sublattice \mathcal{L}_1 and a semi-sublattice \mathcal{L}_2 . Then given a poset Π of \mathcal{L} and a membership oracle, which determines if a matching of \mathcal{L} is in \mathcal{L}_1 or not, the algorithm returns a bouquet defining \mathcal{L}_1 .

FINDBOUQUET(Π):
Input: A poset Π .
Output: A set E of edges defining \mathcal{L}_1 .

1. Initialize: Let $S = \Pi, E = \emptyset$.
2. If M_z is in \mathcal{L}_1 : go to Step 3. Else: $r = t$, go to Step 5.
3. $r = \text{FINDNEXTTAIL}(\Pi, S)$.
4. If r is not NULL: Go to Step 5. Else: Go to Step 7.
5. $F_r = \text{FINDFLOWER}(\Pi, S, r)$.
6. Update:
 - a. For each $u \in F_r$: $E \leftarrow E \cup \{ru\}$.
 - b. $S \leftarrow S \setminus \bigcup_{u \in F_r \cup \{r\}} J'_u$.
 - c. Go to Step 3.
7. Return E .

■ **Figure 4** Algorithm for finding a bouquet.

FINDNEXTTAIL(Π, S):
Input: A poset Π , a splitting set S .
Output: The maximal tail vertex in S , or NULL if there is no tail vertex in S .

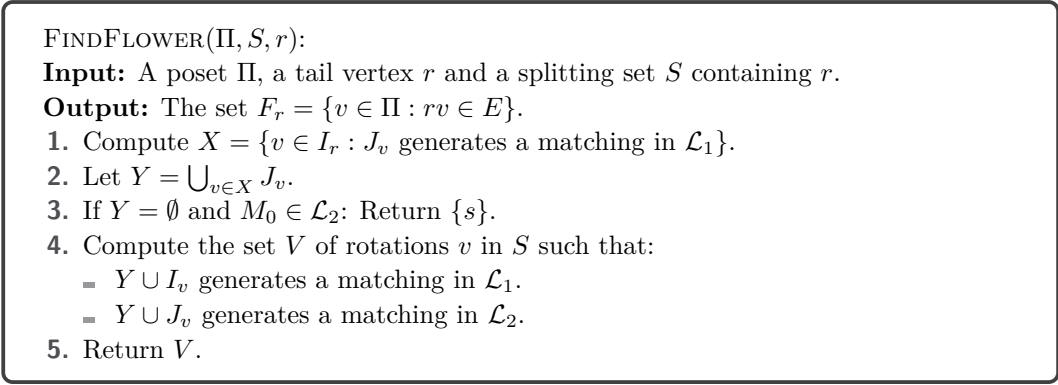
1. Compute the set V of rotations v in S such that:
 - $\Pi \setminus I'_v$ generates a matching in \mathcal{L}_1 .
 - $\Pi \setminus J'_v$ generates a matching in \mathcal{L}_2 .
2. If $V \neq \emptyset$ and there is a unique maximal element v in V : Return v .
 Else: Return NULL.

■ **Figure 5** Subroutine for finding the next tail.

By Theorem 20, the set of tails T_E forms a chain C in Π . The idea of our algorithm, given in Figure 4, is to find the flowers according to their order in C . Specifically, a splitting set S is maintained such that at any point, all flowers outside of S are found. At the beginning, S is set to Π and becomes smaller as the algorithm proceeds. Step 2 checks if M_z is a matching in \mathcal{L}_1 or not. If $M_z \notin \mathcal{L}_1$, the closed subset $\Pi \setminus \{t\}$ separates an edge in E according to Lemma 11. Hence, the first tail on C must be t . Otherwise, the algorithm jumps to Step 3 to find the first tail. Each time a tail r is found, Step 5 immediately finds the flower L_r corresponding to r . The splitting set S is then updated so that S no longer contains L_r but still contains the flowers that have not been found yet. Next, our algorithm continues to look for the next tail inside the updated S . If no tail is found, it terminates.

► **Lemma 26.** *Let v be a rotation in Π . Let $S \subseteq \Pi$ such that both S and $S \cup \{v\}$ are closed subsets. If S generates a matching in \mathcal{L}_1 and $S \cup \{v\}$ generates a matching in \mathcal{L}_2 , v is the head of an edge in E . If S generates a matching in \mathcal{L}_2 and $S \cup \{v\}$ generates a matching in \mathcal{L}_1 , v is the tail of an edge in E .*

► **Lemma 27.** *Given a splitting set S , $\text{FINDNEXTTAIL}(\Pi, S)$ (Figure 5) returns the maximal tail vertex in S , or NULL if there is no tail vertex in S .*



■ **Figure 6** Subroutine for finding a flower.

► **Lemma 28.** *Given a tail vertex r and a splitting set S containing r , FINDFLOWER(Π, S, r) (Figure 6) correctly returns F_r .*

► **Theorem 29 (h).** *FINDBOUQUET(Π), given in Figure 4, returns a set of edges defining \mathcal{L}_1 .*

Proof. From Lemmas 27 and 28, it suffices to show that S is updated correctly in Step 6(b). To be precise, we need that

$$S \setminus \bigcup_{u \in F_r \cup \{r\}} J'_u$$

must still be a splitting set, and contains all flowers that have not been found. This follows from Lemma 23 by noticing that

$$\bigcup_{u \in F_r \cup \{r\}} J'_u = \{v \in \Pi : v \succeq u \text{ for some } u \in R_r\}. \quad \blacktriangleleft$$

Clearly, a sublattice of \mathcal{L} must also be a semi-sublattice. Therefore, FINDBOUQUET can be used to find a canonical path described in Section 4. The same algorithm can be used to check if $M_A \cap M_B = \emptyset$. Let E be the edge set given by the FINDBOUQUET algorithm and H_E be the corresponding graph obtained by adding E to the Hasse diagram of the original rotation poset Π of \mathcal{L}_A . If H_E has a single strongly connected component, the compression Π' has a single meta-element and represents the empty lattice.

7 Finding a Fully Robust Stable Matching

Consider the setting given in the Introduction, with S being the domain of errors, one of which is introduced in instance A . We show how to use the algorithm in Section 6 to find the poset generating all fully robust matchings w.r.t. S . We then show how this poset can yield a fully robust matching that maximizes, or minimizes, a given weight function.

7.1 Studying semi-sublattices is necessary and sufficient

Let A be a stable matching instance, and B be an instance obtained by permuting the preference list of one worker or one firm. Lemma 30 gives an example of a permutation so that $\mathcal{M}_{A \setminus B}$ is not a sublattice of \mathcal{L}_A , hence showing that the case studied in Section 4

1		b	a	c	d	1		c	a	b	d	a		1	2	3	4
2		a	b	c	d	2		a	b	c	d	b		2	1	3	4
3		d	c	a	b	3		d	c	a	b	c		3	1	4	2
4		c	d	a	b	4		c	d	a	b	d		4	3	1	2
firms' preferences in A						firms' preferences in B						workers' preferences in both instances					

■ **Figure 7** An example in which $\mathcal{M}_{A \setminus B}$ is not a sublattice of \mathcal{L}_A .

does not suffice to solve the problem at hand. On the other hand, for all such instances B , Lemma 31 shows that $\mathcal{M}_{A \setminus B}$ forms a semi-sublattice of \mathcal{L}_A and hence the case studied in Section 5 does suffice.

The next lemma pertains to the example given in Figure 7, in which the set of workers is $\mathcal{B} = \{a, b, c, d\}$ and the set of firms is $\mathcal{G} = \{1, 2, 3, 4\}$. Instance B is obtained from instance A by permuting firm 1's list.

► **Lemma 30.** *There exist stable matching instances A and B differing by one agent's preference list such that $\mathcal{M}_{A \setminus B}$ is not a sublattice of \mathcal{L}_A .*

► **Lemma 31.** *For any instance B obtained by permuting the preference list of one worker or one firm, $\mathcal{M}_{A \setminus B}$ forms a semi-sublattice of \mathcal{L}_A .*

► **Proposition 32.** *A set of edges defining the sublattice \mathcal{L}' , consisting of matchings in $\mathcal{M}_A \cap \mathcal{M}_B$, can be computed in polynomial time.*

7.2 Proof of Theorem 2

In this section, we will prove Theorem 2 as well as a slight extension; the latter uses ideas from [18]. Let B_1, \dots, B_k be polynomially many instances in the domain $D \subset T$, as defined in the Introduction. Let E_i be the set of edges defining $\mathcal{M}_A \cap \mathcal{M}_{B_i}$ for all $1 \leq i \leq k$. By Corollary 7, $\mathcal{L}' = \mathcal{M}_A \cap \mathcal{M}_{B_1} \cap \dots \cap \mathcal{M}_{B_k}$ is a sublattice of \mathcal{L}_A .

► **Lemma 33.** *$E = \bigcup_i E_i$ defines \mathcal{L}' .*

Proof. By Lemma 11, it suffices to show that for any closed subset I , I does not separate an edge in E iff I generates a matching in \mathcal{L}' .

I does not separate an edge in E iff I does not separate any edge in E_i for all $1 \leq i \leq k$ iff the matching generated by I is in $\mathcal{M}_A \cap \mathcal{M}_{B_i}$ for all $1 \leq i \leq k$ by Lemma 11. ◀

By Lemma 33, a compression Π' generating \mathcal{L}' can be constructed from E as described in Section 3.1. By Proposition 32, we can compute each E_i , and hence, Π' in polynomial time. Clearly, Π' can be used to check if a fully robust stable matching exists. To be precise, a fully robust stable matching exists iff there exists a proper closed subset of Π' . This happens iff s and t belong to different meta-rotations in Π' , an easy to check condition. Hence, we have Theorem 2.

7.3 Finding maximum weight fully robust stable matchings

We can use Π' to obtain a fully robust stable matching M maximizing $\sum_{wf \in M} W_{wf}$ by applying the algorithm of [19]. Specifically, let $H(\Pi')$ be the Hasse diagram of Π' . Then each pair wf for $w \in \mathcal{W}$ and $f \in \mathcal{F}$ can be associated with two vertices u_{wf} and v_{wf} in $H(\Pi')$ as follows:

- If there is a rotation r moving w to f , u_{wf} is the meta-rotation containing r . Otherwise, u_{wf} is the meta-rotation containing s .
- If there is a rotation r moving w from f , v_{wf} is the meta-rotation containing r . Otherwise, v_{wf} is the meta-rotation containing t .

By Lemma 3 and the definition of compression, $u_{wf} \prec v_{wf}$. Hence, there is a path from u_{wf} to v_{wf} in $H(\Pi')$. We can then add weights to edges in $H(\Pi')$, as stated in [19]. Specifically, we start with weight 0 on all edges and increase weights of edges in a path from u_{wf} to v_{wf} by w_{wf} for all pairs wf . A fully robust stable matching maximizing $\sum_{wf \in M} W_{bwf}$ can be obtained by finding a maximum weight ideal cut in the constructed graph. An efficient algorithm for the latter problem is given in [19].

8 Discussion

The primary focus of this paper is the study of “nearby” stable matching instances where a single agent permutes their preference list. A number of new questions arise: give a polynomial time algorithm for the problem mentioned in the Introduction, of finding a robust stable matching as defined in [19] – given a probability distribution on the domain of errors – even when the error is an arbitrary permutation; and extend to the stable roommate problem and incomplete preference lists [15, 20], as well as popular matchings [10, 16].

Next, we give a hypothetical setting to show potential application of our work to the issue of incentive compatibility. Let A be an instance of stable matching over n workers and n firms. Assume that all $2n$ agents have a means of making their preference lists public simultaneously and a dominant firm, say f , is given the task of computing and announcing a stable matching. Once the matching is announced, all agents can verify that it is indeed stable. It turns out that firm f can cheat and improve its match as follows: f changes its preference list to obtain instance B which is identical to A for all other agents, and computes a matching that is stable for A as well as B using Theorem 2. The other agents will be satisfied that this matching is indeed stable for instance A and f ’s cheating may go undetected.

Finally, considering the number of new and interesting matching markets being defined on the Internet, e.g., see [13], it will not be surprising if new, deeper structural facts about stable matching lattices find suitable applications. For this reason, the problem initiated in [18], which appears to be a fundamental one, deserves further work. In particular, we leave the question of extending our work to the case when the two instances A and B are not nearby but arbitrary, i.e., when multiple agents simultaneously change their preference lists.

References

- 1 Atila Abdulkadiroğlu, Parag A Pathak, and Alvin E Roth. Strategy-proofness versus efficiency in matching with indifferences: Redesigning the NYC high school match. *American Economic Review*, 99(5):1954–78, 2009.
- 2 Atila Abdulkadiroğlu, Parag A Pathak, Alvin E Roth, and Tayfun Sönmez. The boston public school match. *American Economic Review*, 95(2):368–371, 2005.

- 3 Atila Abdulkadiroğlu and Tayfun Sonmez. School choice: A mechanism design approach. *American economic review*, 93(3):729–747, 2003.
- 4 H. Aziz, P. Biro, T. Fleiner, S. Gaspers, R. de Haan, N. Mattei, and B. Rastegari. Stable matching with uncertain pairwise preferences. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 344–352, 2017.
- 5 H. Aziz, P. Biro, S. Gaspers, R. de Haan, N. Mattei, and B. Rastegari. Stable matching with uncertain linear preferences. In *International Symposium on Algorithmic Game Theory*, pages 195–206, 2016.
- 6 A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.
- 7 Garrett Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.
- 8 Charles Blair. Every finite distributive lattice is a set of stable matchings. *Journal of Combinatorial Theory, Series A*, 37(3):353–356, 1984.
- 9 G. C. Calafiore and L. El Ghaoui. On distributionally robust chance-constrained linear programs. *Jour. of Optimization Theory and Applications*, 130(1), December 2006.
- 10 Agnes Cseh, Yuri Faenza, Telikepalli Kavitha, and Vladlena Powers. Understanding popular matchings via stable matchings. *SIAM Journal on Discrete Mathematics*, 36(1):188–213, 2022.
- 11 L. E. Dubins and D. A. Freedman. Machiavelli and the gale-shapley algorithm. *The American Mathematical Monthly*, 88(7):485–494, 1981.
- 12 Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors. *Online and Matching-Based Market Design*. Cambridge University Press, to appear. URL: <https://www.ics.uci.edu/~vazirani/Chapter1.pdf>.
- 13 Simons Institute for the Theory of Computing. Online and matching-based market design, 2019. URL: <https://simons.berkeley.edu/programs/market2019>.
- 14 David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
- 15 Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.
- 16 Telikepalli Kavitha. Matchings, critical nodes, and popular solutions. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 17 Donald Ervin Knuth. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms*. American Mathematical Soc., 1997.
- 18 Tung Mai and Vijay V. Vazirani. Finding stable matchings that are robust to errors in the input. In *European Symposium on Algorithms*, 2018.
- 19 Tung Mai and Vijay V. Vazirani. A natural generalization of stable matching solved via new insights into ideal cuts. In arXiv, 2018.
- 20 David Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.
- 21 Shuichi Miyazaki and Kazuya Okamoto. Jointly stable matchings. *Journal of Combinatorial Optimization*, 38(2):646–665, 2019.
- 22 Alvin E Roth and Marilda Sotomayor. Two-sided matching. *Handbook of game theory with economic applications*, 1:485–541, 1992.
- 23 Richard Stanley. *Enumerative combinatorics*, vol. 1, Wadsworth and Brooks/Cole, Pacific Grove, CA, 1986; second printing, 1996.

A Related Work

The two topics, of stable matching and the design of algorithms that produce solutions that are robust to errors, have been studied extensively for decades and there are today several books on each of them, e.g., see [17, 15, 20] and [9, 6]. Yet, there is a paucity of results at the intersection of these two topics. Indeed, before the publication of [18], we are aware of only two previous works [5, 4]. We remark that the notion of robustness studied in [18] was quite different from that of the previous two works as detailed below.

19:16 A Study of Stable Matching Lattices of “Nearby” Instance

Aziz et al. [5] considered the problem of finding stable matching under uncertain linear preferences. They proposed three different uncertainty models:

1. Lottery Model: Each agent has a probability distribution over strict preference lists, independent of other agents.
2. Compact Indifference Model: Each agent has a single weak preference list in which ties may exist. All linear order extensions of this weak order have equal probability.
3. Joint Probability Model: A probability distribution over preference profiles is specified. They showed that finding the matching with highest probability of being stable is NP-hard for the Compact Indifference Model and the Joint Probability Model. For the very special case that preference lists of one side are certain and the number of uncertain agents of the other side are bounded by a constant, they gave a polynomial time algorithm that works for all three models.

The joint probability model is the most powerful and closest to our setting. The main difference is that in their model, there is no base instance, which is called A in our model. The opportunity of finding new structural results arises from our model precisely because we need to consider two “nearby” instances, namely A and B as described above.

Aziz et al. [4] introduced a pairwise probability model in which each agent gives the probability of preferring one agent over another for all possible pairs. They showed that the problem of finding a matching with highest probability of being stable is NP-hard even when no agent has a cycle in its certain preferences (i.e., the ones that hold with probability 1).

B Proof of Birkhoff’s Theorem using Stable Matching Lattices

Omitted proofs can be found in the *Arxiv version*.

C Other Omitted Proofs

Proof of Lemma 6. It suffices to show that $\mathcal{M}_A \cap \mathcal{M}_B$ is a sublattice of \mathcal{L}_A . Assume $|\mathcal{M}_A \cap \mathcal{M}_B| > 1$ and let M_1 and M_2 be two different matchings in $\mathcal{M}_A \cap \mathcal{M}_B$. Let \vee_A and \vee_B be the join operations under A and B respectively. Likewise, let \wedge_A and \wedge_B be the meet operations under A and B .

By definition of join operation in Section 2.1, $M_1 \vee_A M_2$ is the matching obtained by assigning each worker to its less preferred partner (or equivalently, each firm to its more preferred partner) from M_1 and M_2 according to instance A . Without loss of generality, assume that B is an instance obtained from A by changing the lists of only firms. Since the list of each worker is identical in A and B , its less preferred partner from M_1 and M_2 is also the same in A and B . Therefore, $M_1 \vee_A M_2 = M_1 \vee_B M_2$. A similar argument can be applied to show that $M_1 \wedge_A M_2 = M_1 \wedge_B M_2$.

Hence, $M_1 \vee_A M_2$ and $M_1 \wedge_A M_2$ are both in $\mathcal{M}_A \cap \mathcal{M}_B$ as desired. ◀

Proof of Corollary 7. Assume $|\mathcal{M}_A \cap \mathcal{M}_{B_1} \cap \dots \cap \mathcal{M}_{B_k}| > 1$ and let M_1 and M_2 be two different matchings in $\mathcal{M}_A \cap \mathcal{M}_{B_1} \cap \dots \cap \mathcal{M}_{B_k}$. Therefore, M_1 and M_2 are in $\mathcal{M}_A \cap \mathcal{M}_{B_i}$ for each $1 \leq i \leq k$. By Proposition 6, $\mathcal{M}_A \cap \mathcal{M}_{B_i}$ is a sublattice of \mathcal{L}_A . Hence, $M_1 \vee_A M_2$ and $M_1 \wedge_A M_2$ are in $\mathcal{M}_A \cap \mathcal{M}_{B_i}$ for each $1 \leq i \leq k$. The claim then follows. ◀

Proof of Proposition 10. Let Π' be a compression of Π obtained using the first definition. Clearly, for each meta-rotation in Π' , we can add edges to Π so the strongly connected component created is precisely this meta-rotation. Any additional precedence relations introduced among incomparable meta-rotations can also be introduced by adding appropriate edges.

The other direction is even simpler, since each strongly connected component can be defined to be a meta-rotation and extra edges added can also be simulated by introducing new precedence constraints. ◀

Proof of Lemma 11. Let Π' be a compression corresponding to \mathcal{L}' . By Theorem 1, the matchings in \mathcal{L}' are generated by eliminating rotations in closed subsets of Π' .

First, assume I separates $uv \in E$. Moreover, assume $M \in \mathcal{L}'$ for the sake of contradiction, and let I' be the closed subset of Π' corresponding to M . Let U and V be the meta-rotations containing u and v respectively. Notice that the sets of rotations in I and I' are identical. Therefore, $V \in I'$ and $U \notin I'$. Since $uv \in E$, there is an edge from U to V in H' . Hence, I' is not a closed subset of Π' .

Next, assume that I does not separate any $uv \in E$. We show that the rotations in I can be partitioned into meta-rotations in a closed subset I' of Π' . If I cannot be partitioned into meta-rotations, there must exist a meta-rotation A such that $A \cap I$ is a non-empty proper subset of A . Since A consists of rotations in a strongly connected component of H_E , there must be an edge uv from $A \setminus I$ to $A \cap I$ in H_E . Hence, I separates uv . Since I is a closed subset, uv can not be an edge in H . Therefore, $uv \in E$, which is a contradiction. It remains to show that the set of meta-rotations partitioning I is a closed subset of Π' . Assume otherwise, there exist meta-rotation $U \in I'$ and $V \notin I'$ such that there exists an edge from U to V in H' . Therefore, there exists $u \in U$, $v \in V$ and $uv \in E$, which is a contradiction. ◀

Proof of Lemma 16. Let R denote the set of vertices reachable from t by a path of edges in E_1 and E_2 . Assume by contradiction that R does not contain s . Consider the matching M generated by rotations in $\Pi \setminus R$. Without loss of generality, assume that $M \in \mathcal{L}_1$. By Lemma 11, $\Pi \setminus R$ separates an edge $uv \in E_2$. Therefore, $u \in R$ and $v \in \Pi \setminus R$. Since $uv \in E_2$, v is also reachable from t by a path of edges in E_1 and E_2 . ◀

Proof of Lemma 17. A closed subset separating $r_{i-1}r_i$ must separate an edge in Q_i . Moreover, any closed subset must separate exactly one of $r_0r_1, \dots, r_{k-2}r_{k-1}, r_{k-1}r_k$. Therefore, the set of closed subsets separating an edge in E_1 (or E_2) remains unchanged. ◀

Proof of Lemma 18. Let e be an edge in $E_1 \cup E_2$ but not in Q . Suppose that $e \in E_1$. Let I be a closed subset separating e . By Lemma 11, the matching generated by I belongs to \mathcal{L}_2 . Since e is not in Q and Q is a path from t to s , I must separate another edge e' in Q . By Lemma 11, I can not separate edges in both E_1 and E_2 . Therefore, e' must also be in E_1 . Hence, the matching generated by I will still be in \mathcal{L}_2 after removing e from E_1 . The argument applies to all closed subsets separating e . ◀

Proof of Lemma 21. Suppose there are at least two maximal rotations u_1, u_2, \dots, u_k ($k \geq 2$) in $T_E \cap S$. Let v_1, \dots, v_k be the heads of edges containing u_1, u_2, \dots, u_k . For each $1 \leq i \leq k$, let $S_i = J_{u_i} \cup J_{v_j}$ where j is any index such that $j \neq i$. Since u_i and u_j are incomparable, $u_j \notin J_{u_i}$. Moreover, $u_j \notin J_{v_j}$ by Lemma 14. Therefore, $u_j \notin S_i$. It follows that S_i contains u_i and separates u_jv_j . Since S_i separates $u_jv_j \in E$, the matching generated by S_i is in \mathcal{L}_2 according to Lemma 11.

Since $\bigcup_{i=1}^k S_i$ contains all maximal rotations in $T_E \cap S$ and S does not separate any edge in E , $\bigcup_{i=1}^k S_i$ does not separate any edge in E either. Therefore, the matching generated by $\bigcup_{i=1}^k S_i$ is in \mathcal{L}_1 , and hence not in \mathcal{L}_2 . This contradicts the fact that \mathcal{L}_2 is a join semi-sublattice. ◀

Proof of Lemma 22. We will show that the set of closed subsets separating an edge in E remains unchanged.

19:18 A Study of Stable Matching Lattices of “Nearby” Instance

Let I be a closed subset separating uv . Then I must also separate rv since $r \succ v$.

Now suppose I is a closed subset separating rv . We consider two cases:

- If $u \in I$, I must contain x since $u \succ x$. Hence, I separates an edge in the path from r to x .
- If $u \notin I$, I separates uv . ◀

Proof of Lemma 23. The lemma follows from the claims given below:

▷ **Claim 34.** $S \setminus X$ is a closed subset.

Proof. Let v be a rotation in $S \setminus X$ and u be a predecessor of v . Since S is a closed subset, $u \in S$. Notice that if a rotation is in X , all of its successor must be included. Hence, since $v \notin X$, $u \notin X$. Therefore, $u \in S \setminus X$. ◁

▷ **Claim 35.** $S \setminus X$ contains u for each $u \in (T_E \cap S) \setminus R_r$.

Proof. After replacing edges according to Lemma 22, for each $u \in (T_E \cap S) \setminus R_r$ we must have that u does not succeed any $x \in R_r$. Therefore, $u \notin X$ by the definition of X . ◁

▷ **Claim 36.** $(S \setminus X) \cap R_r = \emptyset$.

Proof. Since $R_r \subseteq X$, $(S \setminus X) \cap R_r = \emptyset$. ◁

▷ **Claim 37.** $S \setminus X$ does not separate any edge in E .

Proof. Suppose $S \setminus X$ separates $uv \in E$. Then $u \in X$ and $v \in S \setminus X$. By Claim 2, u can not be a tail vertex, which is a contradiction. ◁

▷ **Claim 38.** $S \setminus X$ does not cross any edge in E .

Proof. Suppose $S \setminus X$ crosses $uv \in E$. Then $u \in S \setminus X$ and $v \in X$. Let J be a closed subset separating uv . Then $v \in J$ and $u \notin J$.

Since $uv \in E$ and $u \in S$, $u \in T_E \cap S$. Therefore, $r \succ u$ by Lemma 21. Since J is a closed subset, $r \notin J$.

Since $v \in X$, $v \succeq x$ for $x \in R_r$. Again, as J is a closed subset, $x \in J$.

Therefore, J separates an edge in the path from r to x in G_r . Hence, all closed subsets separating uv must also separate another edge in E_r . This contradicts the assumption made in Remark 12. ◁

Proof of Lemma 24. We will show that the set of closed subsets separating an edge in E_r and the set of closed subset separating an edge in E'_r are identical.

Consider a closed subset I separating an edge in $rv \in E'_r$. Since $v \in R_r$, I must separate an edge in E in a path from r to v . By definition, that edge is in E_r .

Now let I be a closed subset separating an edge in $uv \in E_r$. Since $uv \in E$, $u \in T_E \cap S$. By Lemma 21, $r \succ u$. Thus, I must also separate $rv \in E'_r$. ◀

Proof of Lemma 26. Suppose that S generates a matching in \mathcal{L}_1 and $S \cup \{v\}$ generates a matching in \mathcal{L}_2 . By Lemma 11, S does not separate any edge in E , and $S \cup \{v\}$ separates an edge $e \in E$. This can only happen if u is the head of e .

A similar argument can be given for the second case. ◀

Proof of Lemma 27. Let r be the maximal tail vertex in S .

First we show that $r \in V$. By Theorem 20, the set of tails of edges in E forms a chain in Π . Therefore $\Pi \setminus I'_r$ contains all tails in S . Hence, $\Pi \setminus I'_r$ does not separate any edge whose tails are in S . Since S is a splitting set, $\Pi \setminus I'_r$ does not separate any edge whose tails are in $\Pi \setminus S$. Therefore, by Lemma 11, $\Pi \setminus I'_r$ generates a matching in \mathcal{L}_1 . By Lemma 14, $\Pi \setminus J'_r$ must separate an edge in E , and hence generates a matching in \mathcal{L}_2 according to Lemma 11.

By Lemma 26, any rotation in V must be the tail of an edge in E . Hence, they are all predecessors of r according to Theorem 20. ◀

Proof of Lemma 28. First we give two crucial properties of the set Y . By Theorem 20, the set of tails of edges in E forms a chain C in Π .

▷ **Claim 39.** Y contains all predecessors of r in C .

Proof. Assume that there is at least one predecessor of r in C , and denote by r' the direct predecessor. It suffices to show that $r' \in Y$. By Theorem 20, there exists a splitting set I such that $R_{r'} \subseteq I$ and $R_r \cap I = \emptyset$. Let v be the maximal element in $C \cap I$. Then v is a successor of all tail vertices in I . It follows that J_v does not separate any edges in E inside I . Therefore, $v \in X$. Since $J_v \subseteq Y$, Y contains all predecessors of r in C . ◀

▷ **Claim 40.** Y does not contain any rotation in F_r .

Proof. Since Y is the union of closed subset generating matching in \mathcal{L}_1 , Y also generates a matching in \mathcal{L}_1 . By Lemma 11, Y does not separate any edge in E . Since $r \notin Y$, Y must not contain any rotation in F_r . ◀

By Claim 1, if $Y = \emptyset$, r is the last tail found in C . Hence, if $M_0 \in \mathcal{L}_2$, s must be in F_r . By Theorem 20, the heads in F_r are incomparable. Therefore, s is the only rotation in C . FINDFLOWER correctly returns $\{s\}$ in Step 3. Suppose such a situation does not happen, we will show that the returned set is F_r .

▷ **Claim 41.** $V = F_r$.

Proof. Let v be a rotation in V . By Lemma 26, v is a head of some edge e in E . Since Y contains all predecessors of r in C , the tail of e must be r . Hence, $v \in F_r$.

Let v be a rotation in F_r . Since Y contains all predecessors of r in C , $Y \cup I_v$ can not separate any edge whose tails are predecessors of r . Moreover, by Theorem 20, the heads in F_r are incomparable. Therefore, I_v does not contain any rotation in F_r . Since Y does not contain any rotation in F_r by the above claim, $Y \cup I_v$ does not separate any edge in E . It follows that $Y \cup I_v$ generates a matching in \mathcal{L}_1 . Finally, $Y \cup J_v$ separates rv clearly, and hence generates a matching in \mathcal{L}_2 . Therefore, $v \in V$ as desired. ◀

Proof of Lemma 30. $M_1 = \{1a, 2b, 3d, 4c\}$ and $M_2 = \{1b, 2a, 3c, 4d\}$ are stable matching with respect to instance A . Clearly, $M_1 \wedge_A M_2 = \{1a, 2b, 3c, 4d\}$ is also a stable matching under A .

In going from A to B , the positions of workers b and c are swapped in firm 1's list. Under B , $1c$ is a blocking pair for M_1 and $1a$ is a blocking pair for M_2 . Hence, M_1 and M_2 are both in $\mathcal{M}_{A \setminus B}$. However, $M_1 \wedge_A M_2$ is a stable matching under B , and therefore is it not in $\mathcal{M}_{A \setminus B}$. Hence, $\mathcal{M}_{A \setminus B}$ is not closed under the \wedge_A operation. ◀

19:20 A Study of Stable Matching Lattices of “Nearby” Instance

Proof of Lemma 31. Assume that the preference list of a firm f is permuted. We will show that $\mathcal{M}_{A \setminus B}$ is a join semi-sublattice of \mathcal{L}_A . By switching the role of workers and firms, permuting the list of a worker will result in $\mathcal{M}_{A \setminus B}$ being a meet semi-sublattice of \mathcal{L}_A .

Let M_1 and M_2 be two matchings in $\mathcal{M}_{A \setminus B}$. Hence, neither of them are in \mathcal{M}_B . In other words, each has a blocking pair under instance B .

Let w be the partner of f in $M_1 \vee_A M_2$. Then w must also be matched to f in either M_1 or M_2 (or both). We may assume that w is matched to f in M_1 .

Let xy be a blocking pair of M_1 under B . We will show that xy must also be a blocking pair of $M_1 \vee_A M_2$ under B . To begin, the firm y must be f since other preference lists remain unchanged. Since xf is a blocking pair of M_1 under B , $x \succ_f^B w$. Similarly, $f \succ_x f'$ where f' is the M_1 -partner of x . Let f'' be the partner of x in $M_1 \vee_A M_2$. Then $f' \geq_x f''$. It follows that $f \succ_x f''$. Since $x \succ_f^B w$ and $f \succ_x f''$, xf must be a blocking pair of $M_1 \vee_A M_2$ under B . ◀

Proof of Proposition 32. We have that \mathcal{L}' and $\mathcal{M}_{A \setminus B}$ partition \mathcal{L}_A , with $\mathcal{M}_{A \setminus B}$ being a semi-sublattice of \mathcal{L}_A , by Lemma 31. Therefore, FINDBOUQUET(Π) finds a set of edges defining \mathcal{L}' by Theorem 29.

By Lemma 4, the input Π to FINDBOUQUET can be computed in polynomial time. Clearly, a membership oracle checking if a matching is in \mathcal{L}' or not can also be implemented efficiently. Since Π has $O(n^2)$ vertices (Lemma 4), any step of FINDBOUQUET takes polynomial time. ◀

D Modified Deferred Acceptance Algorithms

Omitted algorithms and proofs can be found in the *Arxiv version*.

Degree-Restricted Strength Decompositions and Algebraic Branching Programs

Fulvio Gesmundo ✉ 

Saarland University, Saarbrücken, Germany

Purnata Ghosal ✉

University of Warwick, UK

Christian Ikenmeyer ✉

University of Warwick, UK

Vladimir Lysikov ✉ 

QMATH, Department of Mathematical Sciences, University of Copenhagen, Denmark

Abstract

We analyze Kumar’s recent quadratic algebraic branching program size lower bound proof method (CCC 2017) for the power sum polynomial. We present a refinement of this method that gives better bounds in some cases.

The lower bound relies on Noether-Lefschetz type conditions on the hypersurface defined by the homogeneous polynomial. In the explicit example that we provide, the lower bound is proved resorting to classical intersection theory.

Furthermore, we use similar methods to improve the known lower bound methods for slice rank of polynomials. We consider a sequence of polynomials that have been studied before by Shioda and show that for these polynomials the improved lower bound matches the known upper bound.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Lower bounds, Slice rank, Strength of polynomials, Algebraic branching programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.20

Funding *Purnata Ghosal*: DFG IK 116/2-1 and EPSRC EP/W014882/1. The work was done while this author was at the University of Liverpool.

Christian Ikenmeyer: DFG IK 116/2-1 and EPSRC EP/W014882/1. The work was done while this author was at the University of Liverpool.

Vladimir Lysikov: ERC 818761 and VILLUM FONDEN via the QMATH Centre of Excellence (Grant No. 10059).

Acknowledgements We would like to thank Daniele Agostini for many helpful discussions during the development of this work. We also thank Edoardo Ballico, Luca Chiantini, Giorgio Ottaviani and Kristian Ranestad for pointing out useful references on the Noether-Lefschetz property and related results. We thank the anonymous referees for their helpful comments.

1 Introduction

Homogeneous algebraic branching programs are a fundamental machine model for the computation of homogeneous polynomials. Their noncommutative version is completely understood (even in terms of border complexity) since Nisan’s 1991 paper [21]; they are the model of choice in [7] for succinct presentation of a system of homogeneous polynomial equations; and they can be used to phrase Valiant’s famous determinant versus permanent question [26] in a homogeneous way: Does the minimal size of the required homogeneous algebraic branching program for the permanent polynomial grow superpolynomially? Phrasing Valiant’s question in this way removes the “padding” problem in geometric complexity theory, so this is a way of circumventing the GCT no-go results in [18, 8].



© Fulvio Gesmundo, Purnata Ghosal, Christian Ikenmeyer, and Vladimir Lysikov; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 20; pp. 20:1–20:15



Leibniz International Proceedings in Informatics

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

Even though Valiant’s problem is the flagship problem in algebraic complexity theory, only very weak size bounds on homogeneous algebraic branching programs are known. The best lower bound so far was recently proved by Kumar [19] for the power sum polynomial. His proof and recent lower bounds proofs in other algebraic computation models (Chatterjee et al. [10] for general algebraic branching programs, and Kumar and Volk [20] for determinantal complexity) employ decompositions of the form

$$F = \sum_{k=1}^r G_k H_k + R, \tag{1}$$

where F is the polynomial for which the lower bound is proven, and G_k, H_k, R are polynomials such that $\deg R < \deg F$ and $G_k(0) = H_k(0) = 0$. Kumar’s recent $(d - 1)\lceil \frac{n}{2} \rceil$ bound on the size of homogeneous algebraic branching programs can be proven by considering simpler decompositions

$$F = \sum_{k=1}^r G_k H_k, \tag{2}$$

where F is a homogeneous polynomial and G_k, H_k are homogeneous polynomials of degree strictly smaller than F . Decompositions of this form have been investigated in algebraic geometry as well: for instance, in [11, 23], they were used to study rational points on certain algebraic varieties; they appear in [9] to characterize complete intersections contained in a given hypersurface; recently, they were used in [1] to give a proof of Stillman’s conjecture. Following [1], we say that (2) is a *strength decomposition* of the polynomial F ; the minimum r for which a strength decomposition exists is called the *strength* of F . In the literature, the strength of F is also called *Schmidt rank* or *h-invariant*.

In this paper we analyze Kumar’s lower bound proof method. The proof in [19] is tailored to the power sum polynomial, but the method is applicable to every polynomial. We present the general version of Kumar’s technique in Section 3.1. We refine this technique introducing the notion of k -restricted strength of a polynomial and we provide a lower bound for this notion based on geometric properties of the hypersurface defined by the polynomial. These properties are based on the non-existence of subvarieties of low codimension and low degree in the associated hypersurface and can be interpreted as higher codimension versions of a Noether-Lefschetz type condition [16].

We apply the refined method to give a lower bound on the size of a homogeneous algebraic branching program for an explicit family of polynomials

$$P_{n,d}(x_0, \dots, x_{2n}) = x_0^d + \sum_{k=1}^n x_{2k-1} x_{2k}^{d-1};$$

$P_{n,d}$ is a homogeneous polynomial of degree d in $N = 2n + 1$ variables. Kumar’s technique directly applied to this polynomial gives a lower bound $\lceil \frac{N}{4} \rceil (d - 1)$. For $d < 2^{N/4}$ we improve the lower bound by an additive term of approximately $N/2$. If the degree is exponential in N , we get a further additive improvement of order $\frac{N}{2} d^{O(1/N)}$, see Corollary 17(c).

In Section 3.4, we further study the notion of slice rank of homogeneous polynomials, which is a special case of k -restricted strength when $k = 1$. Theorem 18 gives a method to prove lower bounds on the slice rank. Using this result, in Theorem 20 we compute the slice rank of polynomials

$$S_{n,d}(x_0, \dots, x_{n+1}) = \sum_{i=0}^{n-1} x_i x_{i+1}^{d-1} + x_n x_0^{d-1} + x_{n+1}^d.$$

that have been studied by Shioda [24, 25]. For $n = 4$ and 6 we find that the slice rank is equal to $\frac{N}{2} + 1$, where $N = n + 2$ is the number of variables. To the authors' knowledge, this is the first lower bound for the slice rank better than $\lceil N/2 \rceil$. This translates to a lower bound $\frac{N}{2}(d-1) + 2$ on the homogeneous ABP complexity. Again, this is the first lower bound better than Kumar's $\lceil \frac{N}{2} \rceil(d-1)$. We conjecture that these bounds continue to hold for $S_{n,d}$ with arbitrary even n , see Conjecture 22.

2 Preliminaries

We work over the field of complex numbers \mathbb{C} . A homogeneous linear polynomial is called a *linear form*. The ideal generated by polynomials F_1, \dots, F_m is denoted by $\langle F_1, \dots, F_m \rangle$. An ideal is homogeneous if it admits a set of homogeneous generators. Every (homogeneous) ideal in a polynomial ring admits a finite set of (homogeneous) generators [13, Theorem 1.2].

2.1 Projective geometry

Since we work with homogeneous polynomials, it is convenient to work in projective space \mathbb{P}^n , which is defined as $(\mathbb{C}^{n+1} \setminus \{0\})/\mathbb{C}^\times$, that is, points in \mathbb{P}^n correspond to lines through the origin in \mathbb{C}^{n+1} . Given a nonzero vector $v = (v_0, \dots, v_n) \in \mathbb{C}^{n+1}$, write $[v] = (v_0 : \dots : v_n)$ for the corresponding point in \mathbb{P}^n . We refer to [17] for basics of projective geometry and we only record some basic facts. Given a homogeneous ideal $I = \langle F_1, \dots, F_m \rangle$, write $Z(I) = Z(F_1, \dots, F_m) = \{[v] \in \mathbb{P}^n : F_j(v) = 0 \text{ for every } j\}$; a subset $X \subset \mathbb{P}^n$ is a variety if $X = Z(I)$ for some homogeneous ideal I . A variety is irreducible if it is not the proper union of two varieties; every variety X can be written uniquely as a union of finitely many irreducible subvarieties $X = \bigcup_1^r X_j$; X_1, \dots, X_r are called irreducible components of X . We refer to [17, Ch. 11 and Ch. 18] for the definitions and the basic properties of dimension and degree of a variety X , denoted respectively $\dim X$ and $\deg X$. The dimension of \mathbb{P}^n is n . The codimension of $X \subseteq \mathbb{P}^n$ is $\text{codim } X = n - \dim X$; if $X = \emptyset$, $\text{codim } X = n + 1$. A variety X is called *hypersurface* if all its irreducible components have codimension 1; in this case $X = Z(F)$ is defined by a principal ideal $\langle F \rangle$.

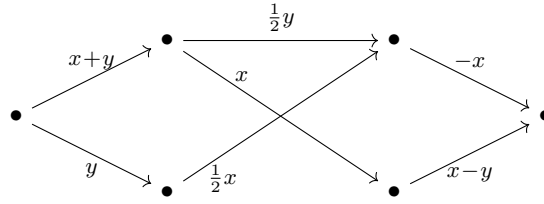
A (*projective*) *linear subspace* in \mathbb{P}^n is a variety defined by linear forms. The codimension of $Z(L_1, \dots, L_r)$, for some linear forms L_1, \dots, L_r , equals the number of linearly independent elements among L_1, \dots, L_r ; in particular $\text{codim } Z(L_1, \dots, L_r) \leq r$. A *line* is a linear subspace of dimension 1. The line spanned by two distinct points $[x], [y] \in \mathbb{P}^n$ is the unique line containing them. This line consists of all points of the form $[\alpha x + \beta y]$ where $(\alpha, \beta) \neq (0, 0)$.

Let $X \subset \mathbb{P}^n$ be a variety. The *projective cone* over X with vertex $p \notin X$ is the union of all lines connecting p with a point in X .

Given a hypersurface $Z(F) \subseteq \mathbb{P}^n$, write $\text{Sing}(F) = Z(\frac{\partial F}{\partial x_i} : i = 0, \dots, n)$, which is a subvariety of $Z(F)$. For example, if $F = x_0^d + x_1^d + x_2^d$, then $\text{Sing}(F) = Z(x_0^{d-1}, x_1^{d-1}, x_2^{d-1}) = \emptyset \subseteq \mathbb{P}^2$; if $F = x_0^d + x_1 x_2^{d-1}$ then $\text{Sing}(F) = Z(x_0^{d-1}, x_2^{d-1}, x_1 x_2^{d-2}) = Z(x_0, x_2) = \{[0 : 1 : 0]\} \subseteq \mathbb{P}^2$. If the factorization of F into irreducible polynomials does not have repeated factors, then $\text{Sing}(F)$ coincides with the *singular locus* of the hypersurface $Z(F)$, see, e.g., in [17, Ch. 14].

We mention the following two fundamental results in algebraic geometry.

► **Theorem 1** (Krull height theorem for polynomial rings [2, Cor. 11.17]). *Let $X \subset \mathbb{P}^n$ be a variety, with $X = Z(F_1, \dots, F_c)$. Then all irreducible components of X have codimension at most c .*



■ **Figure 1** A homogeneous algebraic branching program of size 4, computing $(x + y) \cdot \frac{1}{2}y \cdot (-x) + (x + y) \cdot x \cdot (x - y) + y \cdot \frac{1}{2}x \cdot (-x) = x^3 - x^2y - \frac{3}{2}xy^2$.

► **Theorem 2** (Bezout inequality, see [6, Thm. 8.28]). *If a projective variety $X \subset \mathbb{P}^n$ is cut out by c polynomials of degrees d_1, \dots, d_c , the sum of degrees of its irreducible components is at most $\prod_{i=1}^c d_i$.*

Further, in the proof of Lemma 16, we require some basics of intersection theory. We introduce the required notions and references in Section 3.3.

2.2 Algebraic branching programs

The computational model of algebraic branching programs was first formally defined by Nisan [21] in the context of noncommutative computation, but essentially the same model was used by Valiant in his famous proof of universality of determinant [26]. The computational power of algebraic branching programs is intermediate between the one of general arithmetic circuits and the one of arithmetic formulas. It is a convenient model for algebraic methods, because its power can be captured by restrictions of determinants or iterated matrix multiplication polynomials, which allows for the use of well developed tools from algebra and algebraic geometry. In this paper we only consider homogeneous algebraic branching programs.

► **Definition 3.** *A layered directed graph is a directed graph in which the set of vertices is partitioned into layers indexed by integers so that each edge connects vertices in consecutive layers.*

► **Definition 4.** *A homogeneous algebraic branching program (ABP) in variables x_1, \dots, x_n is a layered directed graph with one source and one sink, and with edges labeled by linear forms in x_1, \dots, x_n . The weight of a path in an ABP is the product of labels on the edges of the path. The polynomial computed between vertices u and v is the sum of the weights of all paths from u to v . The polynomial computed by an ABP is the polynomial computed between the source and the sink.*

The size of an ABP is the number of its inner vertices, namely all vertices except the source and the sink. For a homogeneous polynomial F , its homogeneous ABP complexity $B_{\text{hom}}(F)$ is the minimal size of a homogeneous ABP computing F .

See Figure 1 for an example of an ABP.

2.3 Strength and slice rank of polynomial

► **Definition 5.** *Let F be a homogeneous polynomial of degree d . A strength decomposition of F is a decomposition of the form*

$$F = \sum_{k=1}^r G_k H_k$$

where G_k and H_k are homogeneous polynomials of degree less than d . The strength of F is

$$\text{str}(F) = \min\{r : F \text{ has a strength decomposition with } r \text{ summands}\}.$$

The following is a basic lower bound for the strength of a polynomial. It appears in the introduction of [1] and [3, Remark 4.3]; in [19] it is mentioned with a reference to a personal communication with Saptharishi.

► **Proposition 6.** $\text{str}(F) \geq \lceil \frac{1}{2} \text{codim Sing}(F) \rceil$.

Proof. If $F = \sum_{k=1}^r G_k H_k$, then $\frac{\partial}{\partial x_i} F = \sum_{k=1}^r G_k \frac{\partial}{\partial x_i} H_k + \sum_{k=1}^r H_k \frac{\partial}{\partial x_i} G_k$. Thus all the partial derivatives of F lie in the ideal $\langle G_1, \dots, G_r, H_1, \dots, H_r \rangle$ and therefore the zero set $Z(G_1, \dots, G_r, H_1, \dots, H_r)$ is contained in $\text{Sing}(F)$.

Therefore, applying Theorem 1, we deduce

$$\text{codim Sing}(F) \leq \text{codim } Z(G_1, \dots, G_r, H_1, \dots, H_r) \leq 2r$$

and the required lower bound follows. ◀

► **Remark 7.** The bound of Proposition 6 gives essentially the only known lower bound method for strength which can be applied to explicit polynomials. Different methods are applied to polynomials of a specific form satisfying an unspecified genericity condition: for instance, in [3], it is shown that a polynomial of the form $F = x_1^2 f_1 + x_2^2 f_2 + x_3^2 f_3 + x_4^2 f_4$ with generic f_1, \dots, f_4 has strength 4; this is however achieved using indirect methods [12, 5].

► **Definition 8.** Let F be a homogeneous polynomial of degree d . A slice rank decomposition of F is a decomposition of the form

$$F = \sum_{k=1}^r L_k H_k$$

where L_k are linear forms. The minimal number of summands in a strength decomposition of F is called the slice rank of F and is denoted by $\text{sr}(F)$.

We point out that the notion of slice rank of tensors [22] is related but geometrically very different from the slice rank of homogeneous polynomials defined above.

Clearly, slice rank decompositions are a special class of strength decompositions and thus $\text{sr}(F) \geq \text{str}(F)$. It is known that for generic polynomials the optimal strength decomposition is a slice rank decomposition [4]; in particular $\text{str}(F) = \text{sr}(F)$ for generic F .

Slice rank decompositions of a polynomial F have a clear geometric interpretation in terms of linear subspaces contained in the hypersurface $Z(F)$.

► **Proposition 9.** Let F be a homogeneous polynomial. We have $\text{sr}(F) \leq r$ if and only if $Z(F)$ contains a linear subspace of codimension r .

Proof. The polynomial F admits a decomposition $F = \sum_{k=1}^r L_k H_k$ for linear forms L_1, \dots, L_k if and only if $F \in \langle L_1, \dots, L_k \rangle$. The ideal $\langle L_1, \dots, L_k \rangle$ is radical, in the sense of [13, Sec. 1.6]. Therefore, by the classic Nullstellensatz [13, Thm. 1.6], the condition $F \in \langle L_1, \dots, L_k \rangle$ is equivalent to the condition $Z(F) \supset Z(L_1, \dots, L_r)$. ◀

3 Degree-restricted strength decompositions and lower bounds on ABP size

3.1 Basic properties and connection to ABPs

In this section we introduce the degree-restricted strength decompositions and present a streamlined proof of Kumar's lower bound generalized to arbitrary polynomials based on Proposition 6.

► **Definition 10.** Let F be a homogeneous polynomial of degree d . A strength decomposition $F = \sum_{k=1}^r G_k H_k$ is called j -restricted if $\deg G_k = j$ for all k . The j -restricted strength $\text{str}_j(F)$ is the minimal number of summands in a j -restricted strength decomposition of F .

The following basic properties are clear from the definition.

► **Proposition 11.** Let F be a homogeneous polynomial of degree d and let j be an integer such that $1 \leq j < d$. The following statements hold.

- (a) $\text{str}_j(F) \geq \text{str}(F)$;
- (b) $\text{str}_j(F) = \text{str}_{d-j}(F)$;
- (c) $\text{str}_1(F) = \text{sr}(F)$;

► **Theorem 12.** For every homogeneous polynomial F of degree d

$$B_{\text{hom}}(F) \geq \sum_{j=1}^{d-1} \text{str}_j(F).$$

Proof. Let A be a homogeneous ABP computing F with source s and sink t . Denote by $A[v, w]$ the polynomial computed between vertices v and w . Let V_j be the set of vertices in the j -th layer. Since each path from the source to the sink contains exactly one vertex from each layer, we have

$$F = A[s, t] = \sum_{v \in V_j} A[s, v]A[v, t].$$

If v lies in the j -th layer then $\deg A[s, v] = j$, because each path from s to v contains j edges. Thus F has a j -restricted strength decomposition with $|V_j|$ summands, showing $|V_j| \geq \text{str}_j(F)$. Summing over all layers, we obtain the desired lower bound. ◀

Theorem 12 is similar to Nisan's result for noncommutative ABPs [21, Thm. 1], but in the noncommutative setting the analogue of strength can be easily described as the rank of the partial derivative matrix. In the commutative setting there is no similar characterization of strength.

From Theorem 12, Proposition 11(a) and Proposition 6, we immediately obtain the following B_{hom} lower bounds technique.

► **Corollary 13** (Kumar's singular locus lower bounds technique). For every homogeneous polynomial $F \in \mathbb{C}[x_0, \dots, x_n]$ of degree d

$$B_{\text{hom}}(F) \geq (d-1) \lceil \frac{1}{2} \text{codim Sing}(F) \rceil.$$

In particular, if $\text{Sing}(F) = \emptyset$, then

$$B_{\text{hom}}(F) \geq (d-1) \lceil \frac{n+1}{2} \rceil.$$

The power sum $F = x_0^d + \dots + x_n^d$ of degree d in $n+1$ variables satisfies $\text{Sing}(F) = \emptyset$ and hence a direct application of Corollary 13 gives $B_{\text{hom}}(x_0^d + \dots + x_n^d) \geq (d-1) \lceil \frac{n+1}{2} \rceil$, which recovers Kumar's result [19].

3.2 Lower bound on degree-restricted strength

In this section we prove a lower bound on the degree-restricted strength of polynomials. It is based on the connection between strength decompositions and low degree subvarieties in $Z(F)$, which generalizes Proposition 9. We provide an explicit sequence of polynomials for which we obtain a lower bound that is slightly stronger than the one from Proposition 6.

► **Theorem 14.** *Let $F \in \mathbb{C}[x_0, \dots, x_n]_d$ be a homogeneous polynomial. Suppose that the zero set $Z(F)$ does not contain irreducible subvarieties X with $\text{codim } X \leq c$ and $\text{deg } X < s$ for some $s \geq 2$. Then*

$$\text{sr}(F) \geq c + 1$$

and

$$\text{str}_k(F) \geq \min\{c + 1, \lceil \log_k s \rceil\}$$

Proof. The statement for the slice rank follows from Proposition 9 as $Z(F)$ does not contain linear subspaces, that is, irreducible subvarieties of degree 1, of codimension c .

Assume F has a k -restricted decomposition

$$F = \sum_{j=1}^r G_j H_j.$$

Consider the variety $Y = Z(G_1, \dots, G_r)$. Since F lies in the ideal $\langle G_1, \dots, G_r \rangle$, Y is a subvariety of $Z(F)$. Since Y is defined by r polynomials of degree k , by Theorem 1, the codimension of every irreducible component of Y is at most r ; moreover, by Theorem 2 the sum of degrees of its irreducible components is at most k^r , hence the same holds for each component.

Suppose $r \leq c$. Since $Z(F)$ does not contain subvarieties X with $\text{codim } X \leq c$ and $\text{deg } X < s$, for each irreducible component X of Y we have $k^r \geq \text{deg } X \geq s$, so $r \geq \log_k s$. Since r is an integer, we obtain the lower bound $r \geq \lceil \log_k s \rceil$. Hence, either $r \geq c + 1$ or $r \geq \lceil \log_k s \rceil$, and we conclude $r \geq \min\{c + 1, \lceil \log_k s \rceil\}$ as desired. ◀

► **Remark 15.** In Theorem 14 it suffices to require that $Z(F)$ does not contain subvarieties of codimension exactly c and degree smaller than s . This is a consequence of Bertini's Theorem, see [17, Sec. 18]. Indeed, if X is an irreducible subvariety of $Z(F)$ with $\text{codim } X < c$, let X' be the intersection of X with $c - \text{codim } X$ generic hyperplanes; then $\text{codim } X' = c$ and $\text{deg } X' = \text{deg } X$.

Consider the family of polynomials

$$P_{n,d}(x_0, x_1, \dots, x_{2n}) = x_0^d + \sum_{k=1}^n x_{2k-1} x_{2k}^{d-1}$$

with $d \geq 3$. Note that it is clear from the definition of $P_{n,d}$ that $\text{str}_k(P_{n,d}) \leq n + 1$. Also note that $\text{Sing}(P_{n,d})$ is the linear subspace given by $x_0 = x_2 = x_4 = \dots = x_{2n} = 0$. Its codimension is $n + 1$, so the singular locus lower bound on $B_{\text{hom}}(P_{n,d})$ from Corollary 13 is

$$B_{\text{hom}}(P_{n,d}) \geq (d - 1) \lceil \frac{n+1}{2} \rceil.$$

Corollary 17(c) below will provide an improvement of this lower bound, based on Theorem 14. In order to apply Theorem 14, we need a lower bound on the degree of subvarieties of $Z(P_{n,d})$ of low codimension. This is obtained resorting to an intersection theoretic argument, which is explained in the next section.

3.3 Intersection theory of $Z(P_{n,d})$

This section requires some background in intersection theory, for which we refer to [14] and [15]. We use some facts about Chow groups of a variety, which is one of the fundamental objects studied in intersection theory.

Let X be a variety. Given an integer $a \geq 0$, the Chow group $\text{CH}_a(X)$ is an abelian group associated to the variety X . Every irreducible subvariety $Y \subset X$ of dimension a corresponds to an element $[Y] \in \text{CH}_a(X)$ and these elements generate $\text{CH}_a(X)$. Thus, $\text{CH}_a(X)$ consists of integer linear combinations of irreducible a -dimensional subvarieties of X modulo a certain equivalence relation called *rational equivalence*, the definition of which we do not reproduce here. The elements of $\text{CH}_a(X)$ are called *algebraic cycle classes* of dimension a on X .

If $Y \subset X$ is a subvariety, then every subvariety $Z \subset Y$ is also a subvariety of X . This gives rise to a homomorphism $\iota_*: \text{CH}_a(Y) \rightarrow \text{CH}_a(X)$ sending a cycle class of a subvariety Z of Y to the cycle class of the same variety as a subvariety of X . This homomorphism is called the *pushforward* induced by the inclusion $\iota: Y \hookrightarrow X$. It is a special case of the proper pushforward [15, §1.4].

In addition, we can consider the open set $U = X \setminus Y$ as a variety and its subvarieties. The map j^* sending a cycle class $[Z]$ on X to a cycle class $[Z \cap U]$ on U is well defined. Again, this is a special case of a more general construction of flat pullback [15, §1.7].

Thus for $Y \subset X$ we have $\iota_*: \text{CH}_a(Y) \rightarrow \text{CH}_a(X)$ and $j^*: \text{CH}_a(X) \rightarrow \text{CH}_a(X \setminus Y)$. It is an important fact that these homomorphisms compose to give an exact sequence

$$\text{CH}_a(Y) \rightarrow \text{CH}_a(X) \rightarrow \text{CH}_a(X \setminus Y) \rightarrow 0;$$

in other words, $\ker j^* = \text{im } \iota_*$. This exact sequence is called the *excision exact sequence*, see [15, Prop. 1.8].

For simple varieties the Chow groups can be constructed explicitly. For the projective space \mathbb{P}^n the class in $\text{CH}_a(\mathbb{P}^n)$ of a variety is determined by its degree. More precisely, we have $\text{CH}_a(\mathbb{P}^n) \cong \mathbb{Z}$ where the isomorphism is given by the map $\text{deg}: \text{CH}_a(\mathbb{P}^n) \rightarrow \mathbb{Z}$ sending the class of a subvariety Z to $\text{deg } Z$ (see [14, Thm. 2.1] or [15, Ex. 1.9.3]). A consequence of this is that the degree is well defined for cycle classes on projective varieties. That is, if $X \subseteq \mathbb{P}^n$ is a projective variety and Z is a subvariety of X , then the degree of Z can be read from the cycle class $[Z] \in \text{CH}_a(X)$ by applying the pushforward $\iota_*: \text{CH}_a(X) \rightarrow \text{CH}_a(\mathbb{P}^n)$.

On the other hand, for the affine space the Chow groups $\text{CH}_a(\mathbb{A}^n) = 0$ are trivial for every $a < n$ (see [14, Thm. 1.13] or [15, §1.9]).

Finally, we need a statement which relates the Chow groups of a projective cone X over Y to the Chow groups of Y . Recall that X is a cone over Y (with vertex $p \notin Y$) if X is the union of lines $\langle y, p \rangle$ for $y \in Y$. In this case $\text{CH}_a(Y) \cong \text{CH}_{a+1}(X)$ [15, Ex. 2.6.2]. The isomorphism is given by a map $\alpha: \text{CH}_a(Y) \rightarrow \text{CH}_{a+1}(X)$ which sends a cycle class of a subvariety $Z \subset Y$ to the cycle class of the cone over Z .

► **Lemma 16.** *Let $F \in \mathbb{C}[x_0, \dots, x_n]_d$ be a homogeneous polynomial of degree $d \geq 2$. Set $G = F + x_{n+1}x_{n+2}^{d-1}$. Suppose that the degree of every subvariety $Y \subseteq Z(F) \subseteq \mathbb{P}^n$ with $\dim Y = a$ is divisible by s . Then the degree of every subvariety $Y' \subseteq Z(G) \subseteq \mathbb{P}^{n+2}$ with $\dim Y' = a + 1$ is divisible by s .*

Proof. Let $X = Z(G) \subset \mathbb{P}^{n+2}$ and $Z = Z(F) \subset \mathbb{P}^n$. Let $Y \subset \mathbb{P}^{n+1}$ be the variety given by the equation $F(x_0, \dots, x_n) = 0$ in \mathbb{P}^{n+1} . It consists of all points of the form $[\alpha x + \beta e_{n+1}]$ with $F(x) = 0$, so it is the projective cone over Z with the vertex $[e_{n+1}]$.

The intersection theory statements about projective cones discussed above says that the Chow group $\text{CH}_{a+1}(Y)$ is isomorphic to the Chow group $\text{CH}_a(Z)$. The isomorphism $\alpha: \text{CH}_a(Z) \rightarrow \text{CH}_{a+1}(Y)$ takes the class of a subvariety in Z to the class of the cone over this subvariety.

Let H be the hyperplane given by $x_{n+2} = 0$ in \mathbb{P}^{n+2} . Note that $X \cap H \subset H$ is isomorphic to $Y \subset \mathbb{P}^{n+1}$, so we can identify $X \cap H$ with Y . Let U be the open subset $X \setminus Y$. This subset is an affine variety in $\mathbb{A}^{n+2} = \mathbb{P}^{n+2} \setminus H$ given by the equation $F + x_{n+1} = 0$. Thus $U \cong \text{graph } F \cong \mathbb{A}^{n+1}$. It follows that the Chow group $\text{CH}_{a+1}(U)$ is trivial.

The exactness of the excision exact sequence

$$\text{CH}_{a+1}(Y) \rightarrow \text{CH}_{a+1}(X) \rightarrow \text{CH}_{a+1}(U) \rightarrow 0$$

implies that the inclusion pushforward $\iota^* : \text{CH}_{a+1}(Y) \rightarrow \text{CH}_{a+1}(X)$ is surjective.

Both the cone map $\alpha : \text{CH}_a(Z) \rightarrow \text{CH}_{a+1}(Y)$ and $\iota^* : \text{CH}_{a+1}(Y) \rightarrow \text{CH}_{a+1}(X)$ preserve the degree. Composing them, we obtain a degree-preserving surjective homomorphism from $\text{CH}_a(Z)$ to $\text{CH}_{a+1}(X)$. Since the degree of every dimension a subvariety of Z is divisible by s , the same is true for cycle classes in $\text{CH}_a(Z)$ and therefore, for cycle classes in $\text{CH}_{a+1}(X)$, including dimension $a + 1$ subvarieties of X . ◀

► **Corollary 17.** *If $n \geq 1$ and $d \geq 2$, then*

- (a) $\text{sr}(P_{n,d}) = n + 1$;
 - (b) $\text{str}_k(P_{n,d}) \geq \min\{n + 1, \lceil \log_k d \rceil\}$;
 - (c) $\text{B}_{\text{hom}}(P_{n,d}) \geq (d - 1)\lceil \frac{n+1}{2} \rceil + 2\lfloor \frac{n+1}{2} \rfloor$ for $d \leq 2^{\frac{n+1}{2}}$;
- $$\text{B}_{\text{hom}}(P_{n,d}) \geq (d - 1)\lceil \frac{n+1}{2} \rceil + 2\lfloor \frac{n+1}{2} \rfloor \lfloor d^{\frac{1}{n+1}} \rfloor + \sum_{j=\lfloor d^{\frac{1}{n+1}} \rfloor + 1}^{\lfloor d^{\frac{2}{n+1}} \rfloor} (\lceil \log_j d \rceil - \lceil \frac{n+1}{2} \rceil)$$
- for all
- d
- .

Proof. The polynomial $P_{1,d} = x_0^d + x_1 x_2^{d-1}$ is irreducible, e.g., by the Eisenstein criterion [13, Ex. 18.11] applied to it as an element of $\mathbb{C}[x_1, x_2][x_0]$ with the prime ideal $\langle x_1 \rangle$.

It follows that $Z(P_{1,d})$ is an irreducible variety of degree d in \mathbb{P}^2 , so it does not contain subvarieties of codimension 1 other than itself; in particular, every subvariety of codimension 1 has degree divisible by d . Using Lemma 16 inductively, we obtain that every subvariety X of $Z(P_{n,d})$ with $\text{codim } X = n$ has degree divisible by d . The lower bounds for the slice rank and the restricted strength follow by Theorem 14.

The lower bound for the homogeneous ABP complexity follows by Proposition 11(b) and Theorem 12. If $d \leq 2^{\frac{n+1}{2}}$, we use $\text{str}_1(P_{n,d}) = \text{str}_{d-1}(P_{n,d}) = n + 1$ and $\text{str}_j(P_{n,d}) \geq \lceil \frac{n+1}{2} \rceil$ from Proposition 6 for other j to get

$$\text{B}_{\text{hom}}(P_{n,d}) \geq \sum_{j=1}^{d-1} \text{str}_j(P_{n,d}) \geq 2(n + 1) + (d - 3)\lceil \frac{n + 1}{2} \rceil = (d - 1)\lceil \frac{n + 1}{2} \rceil + 2\lfloor \frac{n + 1}{2} \rfloor$$

For the second bound in (c), we separate the sum $\sum_{j=1}^{d-1} \text{str}_j(P_{n,d})$ into three parts.

For $j \leq d^{\frac{1}{n+1}}$ we have $\text{str}_j(P_{n,d}) = \text{str}_{d-j}(P_{n,d}) \geq n + 1$, for $d^{\frac{1}{n+1}} < j \leq d^{\frac{2}{n+2}}$ we use the lower bound $\text{str}_j(P_{n,d}) = \text{str}_{d-j}(P_{n,d}) \geq \lceil \log_j d \rceil$, and for $d^{\frac{2}{n+2}} < j < d - d^{\frac{2}{n+2}}$ Proposition 6 gives $\text{str}_j(P_{n,d}) \geq \lceil \frac{n+1}{2} \rceil$. ◀

We point out that determining explicit hypersurfaces which do not contain low codimension subvarieties of low degree is an extremely hard problem. It is related to the Noether-Lefschetz Theorem, a classical result which, in particular, implies that if F is a general homogeneous polynomial of degree d , then $Z(F)$ does not contain subvarieties X with $\text{codim } X = 2$ and $\text{deg } X \leq d$. A consequence of [27] is that if F is a general homogeneous polynomial of degree $d \geq 6$ in five variables, then $Z(F) \subseteq \mathbb{P}^4$ does not contain subvarieties X with $\text{codim } X \leq 3$ and $\text{deg}(X) \leq d$. Stronger cohomological results hold for very general hypersurfaces; a series of conjectures and open problems is proposed in [16].

3.4 Slice rank lower bound

In this section, we give examples of polynomials for which we prove a slice rank lower bound stronger than the one induced by the strength lower bound of Proposition 6. In one instance, we show an improved lower bound for a polynomial with $\text{Sing}(F) = \emptyset$.

We define the *Shioda polynomials* of degree $d \geq 3$ in $n + 2$ variables:

$$S_{n,d}(x_0, \dots, x_{n+1}) = \sum_{i=0}^{n-1} x_i x_{i+1}^{d-1} + x_n x_0^{d-1} + x_{n+1}^d.$$

The polynomials $S_{n,d}$ were investigated by Shioda [24, 25] as explicit examples for a cohomological version of the Noether-Lefschetz Theorem in middle dimension.

For even n the decomposition

$$S_{n,d} = \sum_{k=0}^{n/2-1} x_{2k+1} (x_{2k+2}^{d-1} + x_{2k} x_{2k+1}^{d-2}) + x_n \cdot x_0^{d-1} + x_{n+1} \cdot x_{n+1}^{d-1} \quad (3)$$

shows that $\text{sr}(S_{n,d}) \leq \frac{n}{2} + 2$. We prove matching lower bounds for $n = 2$ and $n = 4$; we conjecture that the bound holds for all even values of n .

First consider a modified polynomial

$$\hat{S}_{n,d}(x_1, \dots, x_{n+1}) = \sum_{i=1}^{n-1} x_i x_{i+1}^{d-1} + x_{n+1}^d,$$

which is obtained from $S_{n,d}$ by setting $x_0 = 0$. It is easy to verify that $\text{Sing}(\hat{S}_{n,d})$ coincides with the point $[e_{n-1}] = (0 : \dots : 1 : 0 : 0)$; here we use homogeneous coordinates $(x_1 : \dots : x_{n+1})$ on \mathbb{P}^n .

If n is even, then $\text{codim} \text{Sing}(\hat{S}_{n,d}) = n$ and Proposition 6 gives a lower bound $\text{sr}(\hat{S}_{n,d}) \geq \frac{n}{2}$. An analog of (3) gives the upper bound $\frac{n}{2} + 1$. We provide a matching lower bound, relying on the following result which improves the lower bound of Proposition 6.

► **Theorem 18.** *Let $F \in \mathbb{C}[x_0, \dots, x_n]$ be a homogeneous polynomial with $\text{codim} \text{Sing}(F) = s$ even. Then $\text{sr}(F) = \frac{s}{2}$ if and only if there is a linear space $Q \subset \mathbb{Z}(F)$ of codimension $\frac{s}{2}$ containing one of the irreducible components of $\text{Sing}(F)$.*

Proof. If $Q \subset \mathbb{Z}(F)$ is a linear space with $\text{codim} Q = s/2$, then $\text{sr}(F) = s/2$ by Proposition 9.

Conversely, suppose $\text{sr}(F) = s/2$ and let $F = \sum_{k=1}^{s/2} L_k H_k$ be a minimal slice rank decomposition of F . Let $Q = \mathbb{Z}(L_1, \dots, L_{s/2})$, and let $X = \mathbb{Z}(L_1, \dots, L_{s/2}, H_1, \dots, H_{s/2})$. By the minimality of the decomposition, Q is a linear space of codimension $s/2$. Moreover, $X \subseteq Q \subseteq \mathbb{Z}(F)$.

Since X is defined by s polynomials, by Theorem 1 all irreducible components of X have codimension at most s . As in Proposition 6, X is contained in $\text{Sing}(F)$. Therefore, for every irreducible component X' of X , we have $s = \text{codim} \text{Sing}(F) \leq \text{codim} X \leq \text{codim} X' \leq s$. This shows that X' and $\text{Sing}(F)$ have the same dimension, therefore X' is an irreducible component of $\text{Sing}(F)$. Since $X' \subseteq X \subseteq Q$, we conclude. ◀

► **Lemma 19.** *If $d \geq 3$ and n is even, then $\text{sr}(\hat{S}_{n,d}) = \frac{n}{2} + 1$*

Proof. If n is even, Proposition 6 gives the lower bound $\text{sr}(\hat{S}_{n,d}) \geq \frac{n}{2}$. We use induction on n to prove that $\text{sr}(\hat{S}_{n,d}) \neq \frac{n}{2}$. If $n = 0$ the statement is clear.

Suppose by contradiction $\text{sr}(\hat{S}_{n,d}) = \frac{n}{2}$. By Theorem 18 there exists a projective linear subspace $Q \subset Z(\hat{S}_{n,d})$ of dimension $\frac{n}{2}$ containing the singular point $[e_{n-1}]$. Let $H = Z(x_{n-1})$ and let $Q' = Q \cap H$. Since Q contains the point $[e_{n-1}]$, which is not in H , we have $\dim Q' = \frac{n}{2} - 1$.

Observe that for every point $(v_1 : \dots : v_n) \in Q'$, we have $v_n = 0$. To see this, fix $[v] \in Q'$ and consider the line spanned by $[v] \in Q'$ and $[e_{n-1}]$; this line is contained in Q , hence in $Z(\hat{S}_{n,d})$. In particular, for every $\alpha, \beta \in \mathbb{C}$, we have

$$0 = \hat{S}_{n,d}(\alpha v + \beta e_{n-1}) = \alpha^d \left(\sum_{k=1}^{n-3} v_k v_{k+1}^{d-1} + v_{n+1}^d \right) + \alpha^{d-1} \beta v_n^{d-1}.$$

Therefore, this expression must be 0 as a polynomial in α, β and in particular $v_n = 0$ because v_n^{d-1} is the coefficient of $\alpha^{d-1} \beta$.

This shows that the existence of a subspace $Q \subset Z(\hat{S}_{n,d})$ of dimension $\frac{n}{2}$ containing $[e_{n-1}]$ implies the existence of a subspace $Q' \subset Z(\hat{S}_{n,d}) \cap Z(x_{n-1}, x_n)$ of dimension $\frac{n}{2} - 1$. Note that substituting $x_{n-1} = x_n = 0$ into $\hat{S}_{n,d}$, one obtains, up to renaming the variables, the polynomial $\hat{S}_{n-2,d}$. Therefore, the existence of Q' implies $\text{sr}(\hat{S}_{n-2,d}) = \frac{n}{2} - 1 = \frac{n-2}{2}$, in contradiction with the induction hypothesis. This concludes the proof. \blacktriangleleft

► **Theorem 20.** *If $d \geq 5$, then $\text{sr}(S_{4,d}) = 4$.*

Proof. Let $\rho: \mathbb{P}^5 \rightarrow \mathbb{P}^5$ be the map defined by $\rho(x_0 : x_1 : \dots : x_5) = (x_4 : x_0 : x_1 : x_2 : x_3 : x_5)$ which cyclically permutes the first 5 coordinates of a point $(x_0 : x_1 : \dots : x_5)$. Note that the hypersurface $Z(S_{4,d})$ is mapped to itself by ρ .

The lower bound given by Proposition 6 is $\text{sr}(S_{4,d}) \geq 3$ and assume by contradiction that equality holds. Then there exists a linear space $Q \subset Z(S_{4,d}) \subset \mathbb{P}^5$ of codimension 3.

First, we prove a series of claims about the plane Q culminating with the claim that Q contains one of the five points $[e_k] = \rho^k[e_0]$, where $e_0 = (1, 0, \dots, 0)$. Then we derive a contradiction with the lower bound for $\hat{S}_{n,d}$.

Let A_0 be the line $Z(x_1, x_3, x_4, x_5) \subseteq \mathbb{P}^5$ and $A_k = \rho^k A_0$. In other words, A_0 is the set of points of the form $(x_0 : 0 : x_2 : 0 : 0 : 0)$ and A_k is obtained by cyclically shifting the first five coordinates.

► **Claim 20.1.** Q intersects $A_0 \cup A_1 \cup A_2$.

Proof. Since $\dim Q = 2$, by [17, Prop. 11.4] it intersects any codimension 2 linear subspace. Let $p = (p_0 : 0 : p_2 : 0 : p_4 : p_5) \in Q \cap Z(x_1, x_3)$ and $q = (q_0 : q_1 : 0 : q_3 : 0 : q_5) \in Q \cap Z(x_2, x_4)$ be two points which lie in the respective intersections.

If $p = q$, then $p = q = [e_0] \in A_0$ and the Claim is verified.

Suppose $p \neq q$. The line joining p and q lies in Q and, therefore, in $Z(S_{4,d})$. Let $\hat{p}, \hat{q} \in \mathbb{C}^6$ be representatives of p and q respectively. We have $S_{4,d}(\alpha \hat{p} + \beta \hat{q}) = 0$ for all values of α and β . Expand this expression as a polynomial in α, β , and consider the coefficients of $\alpha^d, \alpha^{d-2} \beta^2, \alpha^{d-3} \beta^3$; these must be 0, hence

$$\begin{aligned} p_4 p_0^{d-1} + p_5^d &= 0, \\ \binom{d-1}{2} p_4 p_0^{d-3} q_0^2 + \binom{d}{2} p_5^{d-2} q_5^2 &= 0, \\ \binom{d-1}{3} p_4 p_0^{d-4} q_0^3 + \binom{d}{3} p_5^{d-3} q_5^3 &= 0. \end{aligned}$$

20:12 Degree-Restricted Strength Decompositions and Algebraic Branching Programs

Rewrite these equations as

$$\begin{aligned} p_4 p_0^{d-1} &= -p_5^d, \\ (d-2)p_4 p_0^{d-3} q_0^2 &= -d p_5^{d-2} q_5^2, \\ (d-3)p_4 p_0^{d-4} q_0^3 &= -d p_5^{d-3} q_5^3. \end{aligned}$$

If $p_5 \neq 0$, then dividing the equations by $p_4 p_0^{d-1} = -p_5^d$, we obtain

$$\begin{aligned} (d-2) \left(\frac{q_0}{p_0} \right)^2 &= d \left(\frac{q_5}{p_5} \right)^2 \\ (d-3) \left(\frac{q_0}{p_0} \right)^3 &= d \left(\frac{q_5}{p_5} \right)^3 \end{aligned}$$

from which we have $q_0 = q_5 = 0$, which implies $q \in A_1$. If, on the other hand, $p_5 = 0$, then either $p_0 = 0$ and $p \in A_2$, or $p_4 = 0$ and $p \in A_0$. \triangleleft

► **Claim 20.2.** If $Q \cap A_0 \neq \emptyset$ and $Q \cap A_2 \neq \emptyset$, then Q contains $[e_0]$, $[e_2]$, or $[e_4]$.

Proof. Let p be a point in $Q \cap A_0$ and $q \in Q \cap A_2$, so $p = (p_0 : 0 : p_2 : 0 : 0 : 0)$ and $q = (0 : 0 : q_2 : 0 : q_4 : 0)$. If $p = q$, then they are equal to $[e_2]$. If $p \neq q$, then there is a point on the line that they span which has zero second coordinate. Let this linear combination be $r = (r_0 : 0 : 0 : 0 : r_4 : 0)$. Since $r \in Q \subset Z(S_{4,n})$, we have $S_{4,n}(r) = r_4 r_0^{d-1} = 0$, so r is either $[e_0]$ or $[e_4]$. \triangleleft

► **Claim 20.3.** Q contains one of the five points $[e_k]$.

Proof. Let $S = \{k \mid Q \cap \rho^k A_0 \neq \emptyset\}$. Since $\rho^5 = \text{id}$, we can see S as a subset of $\mathbb{Z}/5\mathbb{Z}$. Claim 20.1 implies that S contains at least one element of $\{0, 1, 2\}$. Because of the cyclic symmetry, an analogous statement is true for every three consecutive values in $\mathbb{Z}/5\mathbb{Z}$. Similarly, cyclically shifted versions of Claim 20.2 imply that if S contains k and $k+2$, then Q contains one of the five basis points.

Without loss of generality, $0 \in S$. If $2 \in S$ or $3 \in S$, then Claim 20.2 guarantees Q contains one of the five basis points. If both $2, 3 \notin S$, then Claim 20.1 applied to the consecutive triples $\{1, 2, 3\}$ and $\{2, 3, 4\}$ implies that $1, 4 \in S$. Since they differ by two, Claim 20.2 applies and we conclude. \triangleleft

By shifting Q cyclically we can assume that $[e_1] \in Q$.

► **Claim 20.4.** If $[e_1] \in Q$, then Q lies in the hyperplane $Z(x_0)$.

Proof. Note that if the line spanned by two points $[v], [w]$ lies in a hypersurface $Z(F)$, then $\sum \frac{\partial F}{\partial x_i}(v)w_j = 0$. Indeed, the function $f(t) = F(\alpha + tw)$ is identically 0 and so is its derivative in t at $t = 0$. By the chain rule, we obtain the desired equality. In particular, for every two points $p, q \in Q$, we obtain $\sum \frac{\partial S_{4,n}}{\partial x_i}(p)q_i = 0$. Fixing $p = [e_1]$, this guarantees $q_0 = 0$ for every $q \in Q$, showing $Q \subseteq Z(x_0)$. \triangleleft

We deduce that Q is contained in $Z(S_{4,n}) \cap Z(x_0) = Z(S_{4,n}, x_0) = Z(\hat{S}_{4,n}, x_0)$. Therefore $Q \subset Z(\hat{S}_{4,n})$ when regarded as a hypersurface in $Z(x_0) = \mathbb{P}^n$. By Proposition 9, this implies $\text{sr}(\hat{S}_{4,n}) \leq 2$, in contradiction with Lemma 19. This contradiction completes the proof. \blacktriangleleft

As a corollary we obtain a lower bound for homogeneous ABP size for Shioda's polynomials in six variables, which improves on Kumar's lower bound for a polynomial with the same number of variables.

► **Corollary 21.** $B_{\text{hom}}(S_{4,d}) \geq 3(d-1) + 2$.

Proof. We have $\text{str}_k(S_{4,n}) \geq 3$ from Proposition 6 and $\text{str}_{d-1} = \text{str}_1(S_{4,n}) = \text{sr}(S_{4,n}) = 4$ from Theorem 20. Using Theorem 12 we get the required lower bound. ◀

A similar (but easier) argument can be used to prove $\text{sr}(S_{2,d}) = 3$. It follows that $B_{\text{hom}}(S_{2,d}) \geq 2(d-1) + 2$. We conjecture that this can be generalized to all Shioda polynomials.

► **Conjecture 22.** For n even we have $\text{sr}(S_{n,d}) = \frac{n}{2} + 2$ and, consequently,

$$B_{\text{hom}}(S_{n,d}) \geq \frac{n+2}{2}(d-1) + 2.$$

4 Geometry of algebraic branching programs

We have seen that ABPs are related to degree-restricted strength decompositions, and degree-restricted strength decompositions are closely related to subvarieties of the hypersurface defined by the computed polynomial. One can also connect existence of ABPs to subvarieties directly.

► **Theorem 23.** Let F be a homogeneous polynomial of degree d . F is computed by a homogeneous ABP with w_k vertices in layer k if and only if there exists a chain of ideals

$$I_1 \supset I_2 \supset \cdots \supset I_{d-1} \supset I_d = \langle F \rangle$$

such that I_k is generated by w_k homogeneous polynomials of degree k .

Proof. Suppose a homogeneous ABP A computes the polynomial F . Recall that we denote the polynomial computed between vertices v and w by $A[v, w]$. Let s and t be the source and the sink of A , and let V_k be the set of vertices in the k -th layer.

Define I_k to be the ideal generated by polynomials $A[s, v]$ for all $v \in V_k$. These polynomials are homogeneous degree k polynomials, because every path from s to $v \in V_k$ has length k . If $w \in V_{k+1}$, then

$$A[s, w] = \sum_{v \in V_k} A[s, v]A[v, w], \tag{4}$$

so all generators of I_{k+1} lie in I_k and thus $I_k \supset I_{k+1}$. The last layer of A contains only the sink, and the corresponding ideal is $\langle F \rangle$.

On the other hand, given a sequence of ideals $I_1 \supset I_2 \supset \cdots \supset I_{d-1} \supset I_d = \langle F \rangle$ such that I_k is generated by homogeneous polynomials of degree k . Let G_{k1}, \dots, G_{kw_k} be the generators of I_k . Since $I_j \supset I_{j+1}$, we have

$$G_{k+1,j} = \sum_{i=1}^{w_k} G_{ki}L_{kij} \tag{5}$$

for some linear forms L_{kij} . Let A be an ABP with w_k vertices in layer k such that the edge from the i -th vertex in the k -th layer to the j -th vertex in the $(k+1)$ -th layer is labeled by L_{kij} . Then the equations (4) coincide with (5) and thus the ABP computes the generator F of the last ideal. ◀

Geometrically, this implies that $Z(F)$ contains a chain of subvarieties $X_1 \subset X_2 \subset \cdots \subset X_{d-1} \subset Z(F)$ where each X_k is cut out by w_k polynomials of degree k .

References

- 1 T. Ananyan and M. Hochster. Small subalgebras of polynomial rings and Stillman’s conjecture. *J. Amer. Math. Soc.*, 33(1):291–309, 2020.
- 2 M. F. Atiyah and I. G. Macdonald. *Introduction to Commutative Algebra*. Addison Wesley, 1969.
- 3 E. Ballico, A. Bik, A. Oneto, and E. Ventura. The set of forms with bounded strength is not closed. *arXiv*, 2020. [arXiv:2012.01237](https://arxiv.org/abs/2012.01237).
- 4 E. Ballico, A. Bik, A. Oneto, and E. Ventura. Strength and slice rank of forms are generically equal, 2021. [arXiv:2102.11549](https://arxiv.org/abs/2102.11549).
- 5 A. Bik. *Strength and noetherianity for infinite tensors*. PhD thesis, Universität Bern, 2020.
- 6 P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften*. Springer-Verlag, Berlin, 1997.
- 7 P. Bürgisser, F. Cucker, and P. Lairez. Rigid continuation paths II. Structured polynomial systems. *arXiv*, 2020. [arXiv:2010.10997](https://arxiv.org/abs/2010.10997).
- 8 P. Bürgisser, C. Ikenmeyer, and G. Panova. No occurrence obstructions in geometric complexity theory. *J. Amer. Math. Soc.*, 32(1):163–193, 2019.
- 9 E. Carlini, L. Chiantini, and A. V. Geramita. Complete intersections on general hypersurfaces. *Michigan Math. J.*, 57:121–136, 2008.
- 10 P. Chatterjee, M. Kumar, A. She, and B. L. Volk. A Quadratic Lower Bound for Algebraic Branching Programs. In *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *LIPICs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 11 H. Davenport and D. J. Lewis. Non-homogeneous cubic equations. *J. London Math. Soc.*, 39:657–671, 1964.
- 12 Jan Draisma. Topological Noetherianity of polynomial functors. *Journal of the American Mathematical Society*, 32(3):691–707, 2019.
- 13 D. Eisenbud. *Commutative Algebra: with a view toward algebraic geometry*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995.
- 14 D. Eisenbud and J. Harris. *3264 and All That - A Second Course in Algebraic Geometry*. Cambridge University Press, Cambridge, 2016.
- 15 W. Fulton. *Intersection theory*, volume 2 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. 3. Folge. A Series of Modern Surveys in Mathematics [Results in Mathematics and Related Areas. 3rd Series. A Series of Modern Surveys in Mathematics]*. Springer-Verlag, Berlin, second edition, 1998.
- 16 P. Griffiths and J. Harris. On the Noether-Lefschetz theorem and some remarks on codimension-two cycles. *Math. Ann.*, 271(1):31–51, 1985.
- 17 J. Harris. *Algebraic geometry. A first course*, volume 133 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1992.
- 18 C. Ikenmeyer and G. Panova. Rectangular Kronecker coefficients and plethysms in geometric complexity theory. *Adv. Math.*, 319:40–66, 2017.
- 19 M. Kumar. A quadratic lower bound for homogeneous algebraic branching programs. *Comput. Complex.*, 28(3):409–435, 2019.
- 20 M. Kumar and B. L. Volk. A Lower Bound on Determinantal Complexity. In *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPICs*, pages 4:1–4:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 21 N. Nisan. Lower bounds for non-commutative computation. In *Proceedings of the 23rd ACM Symp. on Th. of Comp.*, STOC’91, pages 410–418, New York, NY, USA, 1991. ACM.
- 22 W. F. Sawin and T. Tao. Notes on the “slice rank” of tensors, 2016. available at <https://terrytao.wordpress.com/2016/08/24/notes-on-the-slice-rank-of-tensors/>.
- 23 W. M. Schmidt. The density of integer points on homogeneous varieties. *Acta Math.*, 154(3-4):243–296, 1985.

- 24 T. Shioda. On the Picard number of a complex projective variety. *Ann. Sci. École Norm. Sup. (4)*, 14(3):303–321, 1981.
- 25 T. Shioda. Algebraic cycles on a certain hypersurface. In *Algebraic geometry (Tokyo/Kyoto, 1982)*, volume 1016 of *Lecture Notes in Math.*, pages 271–294. Springer, Berlin, 1983.
- 26 Leslie G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979.
- 27 X. Wu. On a conjecture of Griffiths-Harris generalizing the Noether-Lefschetz theorem. *Duke Math. J.*, 60(2):465–472, 1990.

A Simple Polynomial Time Algorithm for Max Cut on Laminar Geometric Intersection Graphs

Utkarsh Joshi

Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Saladi Rahul

Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Josson Joe Thoppil

Department of Information Technology, National Institute of Technology, Karnataka, India

Abstract

In a geometric intersection graph, given a collection of n geometric objects as input, each object corresponds to a vertex and there is an edge between two vertices if and only if the corresponding objects intersect. In this work, we present a somewhat surprising result: a polynomial time algorithm for max cut on *laminar* geometric intersection graphs. In a laminar geometric intersection graph, if two objects intersect, then one of them will completely lie inside the other. To the best of our knowledge, for max cut this is the first class of (non-trivial) geometric intersection graphs with an exact solution in polynomial time. Our algorithm uses a simple greedy strategy. However, proving its correctness requires non-trivial ideas.

Next, we design almost-linear time algorithms (in terms of n) for laminar axis-aligned boxes by combining the properties of laminar objects with vertical ray shooting data structures. Note that the edge-set of the graph is *not* explicitly given as input; only the n geometric objects are given as input.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Graph algorithms analysis

Keywords and phrases Geometric intersection graphs, Max cut, Vertical ray shooting

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.21

1 Introduction

In the *maximum cut* (a.k.a., max cut) problem, the input is an undirected graph, and the goal is to partition the vertex set into two disjoint sets such that the number of edges having their endpoints in different sets is maximized. Max cut is a classical optimization problem. Weighted version of it is in fact one of Karp's original 21 NP-hard problems [24]. Max cut has been studied extensively on several classes of the graphs. It remains NP-hard even for the special classes of the graphs like cubic graphs [5], total graphs [18], chordal graphs, split graphs, co-bipartite graphs, tripartite graphs [7], and permutation graphs (recently shown [14]). As a result, designing approximation algorithms for max cut has naturally received a lot of attention [13, 17, 26, 23]. On the other hand, the classes of graphs which are of relevance in the context of our work are those for which max cut has been shown to be polynomial time solvable such as planar graphs [19], line graphs [18], graphs with bounded treewidth [7], co-bipartite chain graphs [10], graphs not contractible to K_5 [3] and split-indifference graphs [6]. In this work we add another class of graphs to this list.

Geometric intersection graphs

Consider a set $\mathcal{O} = \{O_1, \dots, O_n\}$ which is a collection of n geometric objects (such as intervals and rectangles). In a *geometric intersection graph*, each object corresponds to a vertex and there is an edge between two vertices if and only if the corresponding objects intersect. We



© Utkarsh Joshi, Saladi Rahul, and Josson Joe Thoppil;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 21; pp. 21:1–21:12



Leibniz International Proceedings in Informatics

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

restate the max cut problem in the context of geometric intersection graphs: Color each object in \mathcal{O} either red or green so that the maximum number of pairs in \mathcal{O} *contribute* to the cut. A pair (O_i, O_j) contributes to the cut iff (a) O_i and O_j intersect, and (b) O_i and O_j have different colors.

Max cut on geometric intersection graphs

The complexity of the max cut on interval graphs was unknown for a long time and was mentioned as an open problem way back in 1985 in celebrated paper of Johnson [22]. At SoCG'21, Adhikary et al. [1] showed that max cut is NP-complete *even* for interval graphs. Unit interval graphs are a special case of interval graphs where each interval has unit length. Surprisingly, the complexity status of the max cut on the class of unit interval graphs is still unknown. Two previous results [8, 9] claimed a polynomial time algorithm for unit interval graphs, but they were reported as incorrect later [6, 25]. Recently, in a series of improvements, it was first shown that max cut is NP-complete on interval graphs of interval count four [15] and then for interval count two [4], which is a special case of interval graphs but more general than unit-interval graphs. On the approximation side, a PTAS can be obtained for max cut on unit-interval graphs ([21] shows a PTAS for the *max bisection* problem, but their technique can be adapted for the max cut problem as well). In 2D, max cut on unit-disk graphs [16] has been shown to be NP-hard.

Exact max cut on laminar geometric intersection graphs

The starting point of our work is the following set of observations. Any two general intervals can have two types of intersections: either (a) partially overlap, or (b) one interval lies inside the other. Unit-interval graphs capture the first type of intersection. From the discussion above, it is clear that max cut on unit-interval graphs has received a lot of interest in terms of designing an exact algorithm, or proving its hardness, or the PTAS result. On the other hand, *laminar interval graphs* can capture the second type of intersection, i.e., if two intervals intersect then one of them lies completely inside the other. However, to the best of our knowledge, there has not been any prior work on max cut for laminar interval graphs.

Motivated by this, we study max cut on *laminar geometric intersection graphs*. In a laminar geometric intersection graph, if two objects intersect, then one of the object will lie completely inside the other object (see Figure 1(a) for an example). Somewhat surprisingly, we prove that a polynomial time *exact* algorithm exists for laminar geometric intersection graphs. To the best of our knowledge, for max cut this is the first class of (non-trivial) geometric intersection graphs with an exact polynomial time solution. We propose a simple greedy algorithm to compute the max cut on laminar geometric intersection graphs. However, as is the case with many greedy algorithms, our proof of optimality involves several non-trivial arguments.

Designing a fast algorithm

The following discussion assumes a basic familiarity with computational geometry. Note that the edge-set of the graph is *not* explicitly given as input; only the n geometric objects are given as input. As a result, a fruitful line of research is to exploit the small input size and the geometry of the intersection graph to design almost-linear time (in terms of n) algorithms to compute max cut. As concrete applications, in this work we consider laminar axis-aligned boxes intersection graphs in $3D$ and $2D$. The first step of our algorithm is to compute a *tree representation* of the laminar objects (see Figure 1 for an example). Using range searching

data structures [2] in a standard manner one can construct such a tree representation in $O(n \log^c n)$ time for axis-aligned boxes in 3D for a sufficiently large constant c . Instead, we perform a sweep-plane and use the properties of laminar objects to reduce the original problem to *dynamic 2D vertical ray shooting problem*. As a result, we are able to compute the tree representation in $O\left(n \frac{\log n}{\log \log n}\right)$ expected time for axis-aligned boxes in 3D. Adapting our algorithm for axis-aligned rectangles in 2D leads to an $O(n \log \log U)$ expected time algorithm, where the endpoints of the boxes lie on the integer grid $[U]^2$. The model of computation is the word-RAM model.

2 Laminar geometric intersection graphs

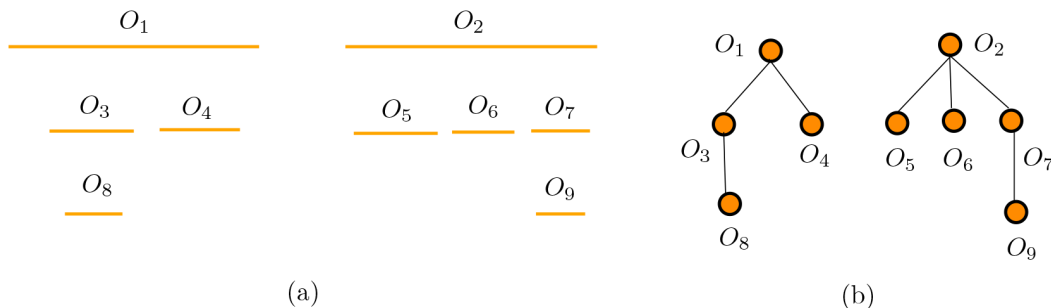
In this section, we will present an exact solution for max cut on laminar geometric intersection graphs. In a laminar geometric intersection graph, if two objects intersect, then one of the objects will lie completely inside the other object (see Figure 1(a) for an example).

2.1 Tree representation and our algorithm

A laminar geometric intersection graph can naturally be represented as a tree. Let $\mathcal{O} = \{O_1, \dots, O_n\}$ be a collection of n laminar geometric objects. An object O_i is said to *dominate* object O_j if and only if O_j lies completely inside O_i . Define level-1 objects of \mathcal{O} to be those objects which are not dominated by anyone in \mathcal{O} . Call them \mathcal{O}_1 . In Figure 1, $\mathcal{O}_1 = \{O_1, O_2\}$. For an integer $i \geq 2$, level- i objects are obtained by computing the set of objects in $\mathcal{O} \setminus \bigcup_{j=1}^{i-1} \mathcal{O}_j$ which are not dominated by any other object in $\mathcal{O} \setminus \bigcup_{j=1}^{i-1} \mathcal{O}_j$. Call them \mathcal{O}_i . In Figure 1, $\mathcal{O}_2 = \{O_3, O_4, O_5, O_6, O_7\}$. We stop the recursion when $\mathcal{O} \setminus \bigcup_{j=1}^{\ell-1} \mathcal{O}_j$ becomes empty, for some integer ℓ .

Now we are ready to describe the tree representation. We will initialise $|\mathcal{O}_1|$ trees, and each object in \mathcal{O}_1 will be made the root of one of the tree (see Figure 1(b)). For $i \in [2, \ell-1]$, each object $O \in \mathcal{O}_i$ will be made the child of that node in level- $(i-1)$ whose corresponding object contains O . See Figure 1. Note that max cut can be computed independently for each tree and then trivially combined to obtain the final solution. As such, without loss of generality we can compute the max cut for a single tree. The following property is easy to observe.

► **Lemma 1.** *Denote the tree representation of \mathcal{O} by \mathcal{T} . Two objects O and O' intersect if and only if O is an ancestor of O' or O' is an ancestor of O in \mathcal{T} .*



■ **Figure 1** (a) An input instance of nine laminar intervals on the real-line. For the sake of clarity, the intervals have been drawn vertically apart, (b) The corresponding tree representation.

21:4 Max Cut on Laminar Geometric Intersection Graphs

From now on we will work with the tree representation \mathcal{T} of \mathcal{O} . Each node in \mathcal{T} will be colored either red or green. In this new terminology, given two nodes $v_i \in \mathcal{T}$ and $v_j \in \mathcal{T}$, the pair (v_i, v_j) *contributes* to the cut iff (a) v_i is an ancestor of v_j , and (b) v_i and v_j have different colors. The goal is to maximize the number of pairs in \mathcal{T} which contribute to the cut.

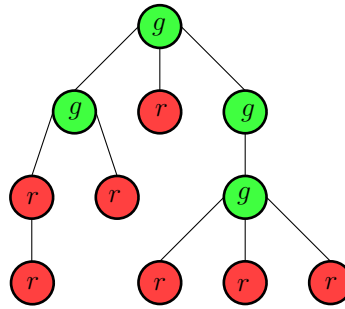
For any node $u \in \mathcal{T}$, let $A(u)$ be the ancestors of u (excluding u) in \mathcal{T} and let $D(u)$ be the nodes in the subtree of u (excluding u).

The algorithm can now be described in a single line:

■ **Algorithm 1** Laminar geometric intersection graphs.

– For each node $u \in \mathcal{T}$, if $|D(u)| \geq |A(u)|$, then color u green; otherwise, color u red.

See Figure 2 for an example. Proving that Algorithm 1 actually reports the maximum cut turns out to be a non-trivial exercise. Our proof will have two steps. First, in Lemma 2 we prove that among all colorings which do not create *inversions* in \mathcal{T} , none of them lead to a larger cut value than the cut value returned by Algorithm 1. A coloring of the nodes in \mathcal{T} is said to create an inversion if there is at least one node which is colored green and its parent node is colored red. Next, in Lemma 3 we prove that among *all* possible ways of coloring \mathcal{T} , it suffices to consider colorings which do not create *inversions* in \mathcal{T} . This proves that Algorithm 1 indeed computes the max cut.



■ **Figure 2** Coloring produced by Algorithm 1 on a tree with eleven nodes. Green colored nodes are labelled g and the red colored nodes are labelled r .

2.2 Optimality of our algorithm

► **Lemma 2.** For any coloring \mathcal{C} , let $Cut(\mathcal{C})$ be the number of pairs in \mathcal{T} contributing to the cut. Let \mathcal{C}_{ni} be any coloring of \mathcal{T} which creates no inversions, and let \mathcal{C}_{alg} be the coloring of \mathcal{T} by our algorithm. Then $Cut(\mathcal{C}_{ni}) \leq Cut(\mathcal{C}_{alg})$.

Proof. We will start with coloring \mathcal{C}_{ni} of the tree and perform n iterations. For $1 \leq i \leq n$, in the i -th iteration, we might potentially *flip* the color of one of the nodes in the tree. Let \mathcal{C}_i be the coloring of the tree at the end of the i -th iteration, for all $1 \leq i \leq n$, and let $\mathcal{C}_0 = \mathcal{C}_{ni}$. We will maintain the following three invariants:

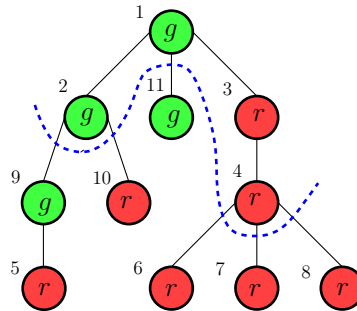
1. $Cut(\mathcal{C}_{i-1}) \leq Cut(\mathcal{C}_i)$, for all $1 \leq i \leq n$, i.e., the cut value does not decrease after each iteration.
2. $\mathcal{C}_n = \mathcal{C}_{alg}$, i.e., at the end of the n iterations, the (potentially) new coloring in the tree will correspond to \mathcal{C}_{alg} .
3. \mathcal{C}_i induces no inversion in \mathcal{T} , for all $1 \leq i \leq n$.

From the first two invariants, it follows that $Cut(\mathcal{C}_{ni}) = Cut(\mathcal{C}_0) \leq Cut(\mathcal{C}_1) \leq Cut(\mathcal{C}_2) \leq \dots \leq Cut(\mathcal{C}_n) = Cut(\mathcal{C}_{alg})$. Now we present the details of the reduction from coloring \mathcal{C}_{ni} to \mathcal{C}_{alg} .

With respect to the coloring \mathcal{C}_{alg} of the tree \mathcal{T} , let \mathcal{G} be the set of nodes in \mathcal{T} colored green. Since \mathcal{C}_{alg} induces no inversion in \mathcal{T} , nodes in \mathcal{G} form a tree (and not a forest). Lets call this tree $\mathcal{T}_{\mathcal{G}}$. The remaining portion of \mathcal{T} , i.e., $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$ will be a collection of trees (a.k.a. a forest) whose nodes are colored red. Unlike the traditional way of either a top-down or a bottom-up traversal of a tree, we will perform a more careful traversal of \mathcal{T} : First, traverse the nodes in $\mathcal{T}_{\mathcal{G}}$ in a top-down manner. Next, traverse the trees in $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$ in a bottom-up manner. See Figure 3. Each iteration corresponds to visiting one of the n nodes in \mathcal{T} .

Top-down traversal of $\mathcal{T}_{\mathcal{G}}$. We will start from the coloring induced by \mathcal{C}_{ni} on tree $\mathcal{T}_{\mathcal{G}}$. We will perform a top-down traversal of $\mathcal{T}_{\mathcal{G}}$ and at the end of the $|\mathcal{G}|$ iterations (i.e., visiting $|\mathcal{G}|$ nodes), all the nodes in $\mathcal{T}_{\mathcal{G}}$ will be colored green.

Let $u \in \mathcal{T}_{\mathcal{G}}$ be the node visited in the i -th iteration. If u is colored green, then no flip operation will be performed, and we will set $\mathcal{C}_i = \mathcal{C}_{i-1}$. The interesting case is when u is colored red, where we will flip the color of u to green. Since \mathcal{C}_{i-1} is a non inversion coloring, all the $|D(u)|$ nodes of \mathcal{T} in the subtree of u will be colored red. Also, due to the top-down traversal of $\mathcal{T}_{\mathcal{G}}$, all the $|A(u)|$ ancestors of u will be colored green. Since Algorithm 1 colored u green, it must be the case that $|D(u)| \geq |A(u)|$. Therefore, flipping the color of u from red to green will ensure that $Cut(\mathcal{C}_i) = Cut(\mathcal{C}_{i-1}) - |A(u)| + |D(u)| \geq Cut(\mathcal{C}_{i-1})$ (satisfying Invariant 1). Also, after the flip operation, the coloring induced by \mathcal{C}_i will have no inversion (satisfying Invariant 3).



■ **Figure 3** Arbitrary \mathcal{C}_{ni} for a tree with eleven nodes. Green colored nodes are labelled g and the red colored nodes are labelled r . The nodes indexed 1 to 4 belong to $\mathcal{T}_{\mathcal{G}}$ and the nodes indexed 5 to 11 belong to $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$. Indexing denotes the order in which the nodes will be traversed.

Bottom-up traversal of $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$. At the end of performing $|\mathcal{G}|$ iterations on $\mathcal{T}_{\mathcal{G}}$, we will have the coloring induced by $\mathcal{C}_{|\mathcal{G}|}$. Now consider any tree in $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$. We will perform a bottom-up traversal of that tree. Let u be a node visited in the i -th iteration. If u is colored red, then no flip operation will be performed and we will set $\mathcal{C}_i = \mathcal{C}_{i-1}$. The interesting case is when u is colored green, where we will flip the color of u to red. Since \mathcal{C}_{i-1} is a no inversion coloring, all the $|A(u)|$ ancestors of u in \mathcal{T} will be colored green. Also, due to bottom-up traversal of the tree, all the $|D(u)|$ nodes in the subtree of u in \mathcal{T} will be colored red. Since Algorithm 1 colored u red, it must be the case that $|D(u)| < |A(u)|$. Therefore, flipping the color of u from green to red will ensure that $Cut(\mathcal{C}_i) = Cut(\mathcal{C}_{i-1}) - |D(u)| + |A(u)| \geq Cut(\mathcal{C}_{i-1})$ (satisfying Invariant 1). Also, after the flip operation, the coloring induced by \mathcal{C}_i will create no inversions (satisfying Invariant 3). Perform the above operations for all the trees in $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$.

At the end of the n iterations, it is easy to see that the coloring in \mathcal{T} corresponds to \mathcal{C}_{alg} : the top-down traversal ensures that all the nodes in \mathcal{G} are colored green and all the nodes in $\mathcal{T} \setminus \mathcal{T}_{\mathcal{G}}$ are colored red, and hence, satisfying Invariant 2. \blacktriangleleft

2.3 A non-inversion coloring of the tree

In this subsection, we will prove the following result.

► **Lemma 3.** *Let \mathcal{C} be an arbitrary coloring of the tree \mathcal{T} and let $Cut(\mathcal{C})$ be the number of pairs in \mathcal{T} contributing to the cut. Then there exists a coloring \mathcal{C}_{ni} which creates no inversions in \mathcal{T} and $Cut(\mathcal{C}_{ni}) \geq Cut(\mathcal{C})$.*

Fix an ordering from left-to-right of the m leaves in \mathcal{T} . The ordering of the leaves naturally fixes an order of all the nodes at each level of \mathcal{T} . For all $1 \leq i \leq m$, let π_i be the root-to-leaf path for the i -th leaf of \mathcal{T} . We start with coloring \mathcal{C} of the tree \mathcal{T} , and let $\mathcal{C}_0 = \mathcal{C}$. We will perform m iterations and in each iteration potentially flip the color of some of the nodes in \mathcal{T} . For all $1 \leq i \leq m$, let \mathcal{C}_i be the coloring of the tree at the end of the i -th iteration. At the end of the i -th iteration, for all $1 \leq i \leq m$, we will ensure that the following invariants are maintained:

1. For all $1 \leq j \leq i$, there are no inversions among nodes on path π_j .
2. $Cut(\mathcal{C}_i) \geq Cut(\mathcal{C}_{i-1})$.

By the first invariant it is guaranteed that the coloring induced by \mathcal{C}_m creates no inversions in \mathcal{T} . Therefore, we set $\mathcal{C}_{ni} = \mathcal{C}_m$. Via the second invariant, it follows that $Cut(\mathcal{C}_{ni}) \geq Cut(\mathcal{C})$. This will prove Lemma 3. For any $1 \leq i \leq m$, we now present the details of the transformation from \mathcal{C}_{i-1} to \mathcal{C}_i . First, we will perform a *modification step*.

Modification steps. At the beginning of the i -th iteration, let $g(\pi_i)$ and $r(\pi_i)$ be the number of green and red colored nodes on path π_i , respectively. We will *modify* the color of the nodes in π_i as follows:

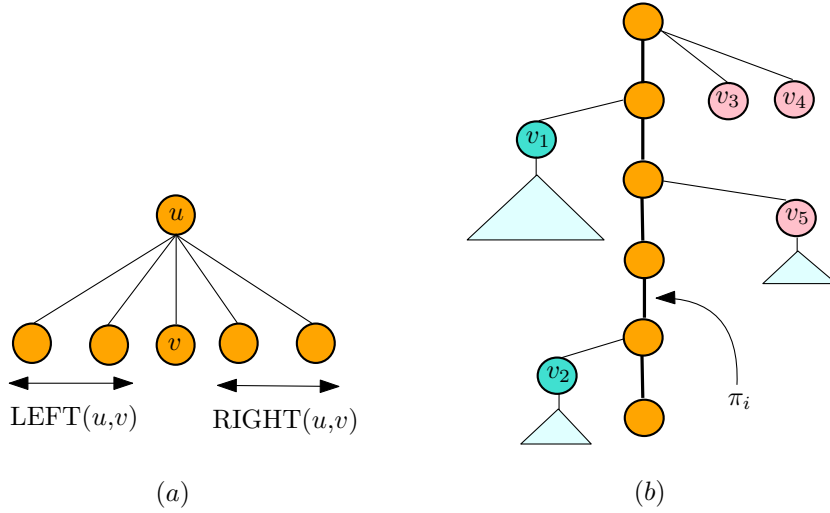
- If $g(\pi_i) \geq r(\pi_i)$, then the first $g(\pi_i)$ nodes on path π_i are colored green, and the remaining $r(\pi_i)$ nodes are colored red.
- Otherwise, the first $r(\pi_i)$ nodes on path π_i are colored green, and the remaining $g(\pi_i)$ nodes are colored red.

Let $E = \{(u, v) | u, v \in \mathcal{T} \text{ and } u \text{ is an ancestor of } v\}$. For any node $u \in \mathcal{T}$, let u_1, u_2, \dots, u_t be its children in the left-to-right ordering. Then we define $LEFT(u, u_j) = \{u_1, u_2, \dots, u_{j-1}\}$ and $RIGHT(u, u_j) = \{u_{j+1}, \dots, u_t\}$. Also, let $\pi_i(1), \pi_i(2), \dots$ be the sequence of nodes on π_i from root-to-leaf. Now we define two sets \mathcal{R}_i and \mathcal{L}_i as follows:

$$\begin{aligned} \mathcal{R}_i &= \{v | v \in RIGHT(\pi_i(j), \pi_i(j+1)) \text{ and } \pi_i(j) \in \pi_i\} \text{ and} \\ \mathcal{L}_i &= \{v | v \in LEFT(\pi_i(j), \pi_i(j+1)) \text{ and } \pi_i(j) \in \pi_i\} \end{aligned}$$

See Figure 4 for an example. Now we will partition E into four subsets E_1, E_2, E_3 , and E_4 . We will argue that the contribution of E_1, E_2, E_3 , and E_4 to the cut after the modifications performed above will not decrease (E_3 will potentially require “flip” operations to ensure its contribution does not go down). This will ensure that $Cut(\mathcal{C}_i) \geq Cut(\mathcal{C}_{i-1})$ (Invariant 2). The four disjoint subsets are defined as follows:

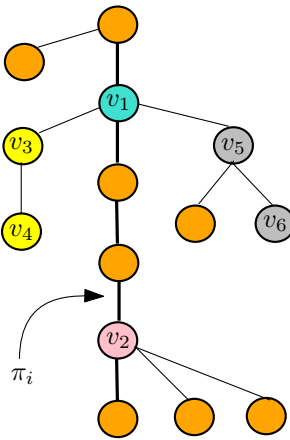
- $E_1 = \{(u, v) | u, v \in \pi_i\}$
- $E_2 = \{(u, v) | w \in \mathcal{L}_i \text{ and } u \in A(w) \text{ and } v \in (D(w) \cup \{w\})\}$



■ **Figure 4** (a) illustrates $\text{LEFT}(u, v)$ and $\text{RIGHT}(u, v)$, and (b) illustrates \mathcal{L}_i and \mathcal{R}_i . In this example, $\mathcal{L}_i = \{v_1, v_2\}$ and $\mathcal{R}_i = \{v_3, v_4, v_5\}$.

- $E_3 = \{(u, v) | w \in \mathcal{R}_i \text{ and } u \in A(w) \text{ and } v \in (D(w) \cup \{w\})\}$
- $E_4 = E \setminus E_1 \setminus E_2 \setminus E_3 = \{(u, v) | w \in \mathcal{L}_i \cup \mathcal{R}_i \text{ and } u, v \in (D(w) \cup \{w\}) \text{ and } v \in A(u)\}$

See Figure 5 for an example. Now we will argue about each subset one after the other.



■ **Figure 5** We illustrate via an example some of the pairs in E_1, E_2, E_3 , and E_4 . For example, $(v_1, v_2) \in E_1, (v_1, v_3) \in E_2, (v_1, v_4) \in E_2, (v_1, v_5) \in E_3, (v_1, v_6) \in E_3, (v_4, v_3) \in E_4, (v_6, v_5) \in E_4$.

► **Lemma 4.** *The number of pairs in E_1 which contribute to the cut does not change after the modification.*

Proof. Before and after the modification, the number of pairs of E_1 which contribute to the cut will be equal to $g(\pi_i) \cdot r(\pi_i)$. ◀

► **Lemma 5.** *For any $v \in \mathcal{R}_i \cup \mathcal{L}_i$, after performing the modification described above, the number of green ancestors will not decrease.*

Proof. For any $v \in \mathcal{R}_i \cup \mathcal{L}_i$, it is obvious from the above definition that all its ancestors lie on π_i . Let $g(v)$ and $r(v)$ be the number of green and red ancestors of v , respectively, *before* performing the modification. Let $g'(v)$ be the number of green ancestors (excluding v) of v *after* performing the modification.

First, consider the case when $g(\pi_i) \geq r(\pi_i)$. If $r(v) + g(v) \leq g(\pi_i)$, then all the ancestors of v after modification will be colored green and hence, $g'(v) = r(v) + g(v) \geq g(v)$; otherwise, the number of ancestors of v after modification colored green will be equal to $g(\pi_i)$ and hence, $g'(v) = g(\pi_i) \geq g(v)$. Therefore, $g'(v) \geq g(v)$.

Next, consider the case when $r(\pi_i) > g(\pi_i)$. If $r(v) + g(v) \leq r(\pi_i)$, then all the ancestors of v after modification will be colored green and hence, $g'(v) = r(v) + g(v) \geq g(v)$; otherwise, the number of ancestors of v after modification colored green will be equal to $r(\pi_i)$ and hence, $g'(v) = r(\pi_i) > g(\pi_i) \geq g(v)$. Therefore, $g'(v) \geq g(v)$. ◀

► **Lemma 6.** *Recall that $E_2 = \{(u, v) | w \in \mathcal{L}_i \text{ and } u \in A(w) \text{ and } v \in (D(w) \cup \{w\})\}$. The contribution of E_2 to the cut does not decrease after the modification.*

Proof. Consider any $w \in \mathcal{L}_i$ and let $E_2(w) = \{(u, v) | u \in A(w) \text{ and } v \in (D(w) \cup \{w\})\}$. First, consider the case where w is colored green before the modification. Then, by invariant 1, since there are no inversions on the path from root to w , all the ancestors of w will be colored green. After the modification, by Lemma 5, it follows that all the ancestors of w will continue to be colored green and hence, the contribution of $E_2(w)$ to the cut does not change.

Next, consider the case where w is colored red before the modification. Then, by invariant 1, all the nodes in $D(w)$ will be colored red. After the modification, by Lemma 5, the number of green ancestors of w will not decrease. Therefore, for any $w \in \mathcal{L}_i$, the contribution of $E_2(w)$ to the cut will not decrease after the modification. ◀

After the modification steps, now we will potentially perform some *flip operations*. The flip operations are needed to handle E_3 .

Flip Operations. For any $w \in \mathcal{R}_i$: if $D(w) \cup \{w\}$ has more green colored nodes than red colored nodes, then *flip* all the colors in $D(w) \cup \{w\}$. Formally, each red (resp., green) colored node is now colored green (resp., red).

► **Lemma 7.** *Recall that $E_3 = \{(u, v) | w \in \mathcal{R}_i \text{ and } u \in A(w) \text{ and } v \in (D(w) \cup \{w\})\}$. The contribution of E_3 to the cut does not decrease after the modification steps and flip operations.*

Proof. Consider any $w \in \mathcal{R}_i$ and let $E_3(w) = \{(u, v) | u \in A(w) \text{ and } v \in (D(w) \cup \{w\})\}$. Because of flip operations, $D(w) \cup \{w\}$ has either equal or more red colored nodes than green colored nodes. Combining this with Lemma 5 (the number of green ancestors will not come down after modification), we conclude that the contribution of $E_3(w)$ to the cut does not go down after modification and flip operations. ◀

► **Lemma 8.** *Recall that $E_4 = E \setminus E_1 \setminus E_2 \setminus E_3 = \{(u, v) | w \in \mathcal{L}_i \cup \mathcal{R}_i \text{ and } u, v \in (D(w) \cup \{w\}) \text{ and } v \in A(u)\}$. The contribution of E_4 to the cut does not change after the modification steps and the flip operations.*

Proof. Consider any edge $(u, v) \in E_4$. We claim that (u, v) will contribute to the cut after the modification if and only if (u, v) contributed to the cut before the modification: if $w \in \mathcal{R}_i$ and nodes in $D(w) \cup \{w\}$ flipped their colors, then the colors of *both* u and v will change. Otherwise, if $w \in \mathcal{R}_i$ and nodes in $D(w) \cup \{w\}$ did not flip their colors, or if $w \in \mathcal{L}_i$, then the colors of u and v do not change. ◀

► **Lemma 9** (Invariant 1). *For all $1 \leq j \leq i$, there are no inversions among nodes on path π_j .*

Proof. It is easy to see that there are no inversions among nodes on path π_i . For any $j < i$, consider the nodes in $\pi_i \cap \pi_j$. Before the modification, there were no inversions on π_j . This implies that either $\pi_i \cap \pi_j$ (a) had all nodes colored green, or (b) had a sequence of green followed by red colored nodes. In case (a), after the modification, all the nodes in $\pi_i \cap \pi_j$ will continue to be colored green. In case (b), after the modification, either all nodes in $\pi_i \cap \pi_j$ will become green, or it will continue to be a sequence of green followed by red colored nodes. In all these cases, there will be no inversions created among nodes on path π_j . ◀

3 A fast implementation

Naively, one can construct the tree representation in $O(n^2)$ time by comparing every pair of objects. In this section, we will construct the tree representation in almost-linear time (in terms of n) for axis-aligned boxes in $3D$ and $2D$. Note that given the tree representation, Algorithm 1 takes only $O(n)$ time: Computing $|D(u)|$, for all $u \in \mathcal{T}$, can be done in $O(n)$ time via bottom-up traversal of \mathcal{T} . Analogously, computing $|A(u)|$, for all $u \in \mathcal{T}$, can be done in $O(n)$ time via a top-down traversal of \mathcal{T} .

Let \mathcal{B} be a collection of n laminar axis-aligned boxes in $3D$. Now we describe our algorithm to compute the tree representation \mathcal{T} for \mathcal{B} . To make the presentation simple, consider a dummy box B_0 which contains all the boxes in \mathcal{B} . Add B_0 to \mathcal{B} (this ensures that B_0 is the root of \mathcal{T}). Any axis-aligned box in $3D$ has six faces on its boundary. Define the *left face* and the *right face* to be the boundaries parallel to the yz -plane.

We will perform a sweep-plane. Consider a plane ℓ parallel to the yz -plane. Starting from $x = -\infty$ we will move the plane ℓ towards $x = +\infty$. The event points will be the left and the right faces on the boundary of each box in \mathcal{B} . Assume that the sweep-plane has reached the x -coordinate x_t , and let ℓ_t be the plane ℓ at x_t . Let $\mathcal{B}_t \subseteq \mathcal{B}$ be the set of boxes intersecting the plane ℓ_t . For every $B \in \mathcal{B}_t$, let $R \leftarrow B \cap \ell_t$, i.e., the projection of B onto the plane ℓ_t which is an axis-aligned rectangle in $2D$.

Along with the sweep-plane, we will maintain a *dynamic vertical ray shooting* data structure. In the *orthogonal* version of dynamic vertical ray shooting problem, the input is a collection of horizontal segments in $2D$, and given a query point q' in $2D$, the goal is to report the first segment which is hit by the upward ray starting from q' . Updates include insertion and deletion of segments. Let \mathcal{D}_t be the instance of the vertical ray shooting data structure when the sweep-plane is at x_t . For each $B \in \mathcal{B}_t$, the top and the bottom segment of R are maintained in \mathcal{D}_t . With the top and the bottom segments of R we will maintain a *label* B and B^{par} , respectively, where B^{par} is the parent of B in \mathcal{T} .

► **Lemma 10.** *Suppose that the next event point in the sweep-plane is the left face of $B \in \mathcal{B}$. Consider any point $q(q_x, q_y, q_z)$ on the left face of B . Let x_t be the x -coordinate of the sweep-plane just before it reaches the left face of B . If B_q is the label associated with the segment reported by the vertical ray shooting query on \mathcal{D}_t with query point $q'(q_y, q_z)$, then B_q is the parent of B in \mathcal{T} .*

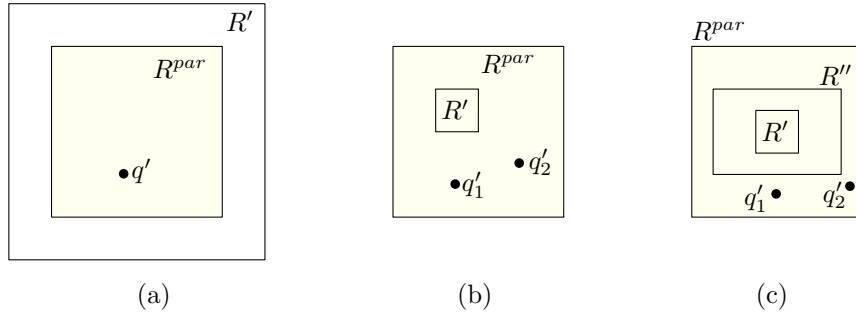
Proof. We will consider the following three cases.

- Since B^{par} contains B , it implies that the rectangle $R^{par} \leftarrow B^{par} \cap \ell_t$ will contain q' . Also, all the ancestors of B^{par} in the tree representation will contain q' . However, by the laminar property, notice that a vertical upward ray from q' will hit the top segment of R^{par} before hitting the top segments of its ancestors. See Figure 6(a).

21:10 Max Cut on Laminar Geometric Intersection Graphs

- Consider any box B' which is a child of B^{par} in the tree representation. Since R' does not contain q' , then a vertical upward ray from q' will either miss the top and the bottom segments of R' , or it will hit the bottom segment of R' before the top segment of R' . See Figure 6(b).
- Consider any box B' which is a descendent of B^{par} , but not a child of B^{par} . Then a vertical upward ray from q' will never hit the bottom or the top segment of B' first. See Figure 6(c).

Based on the above three observations, we conclude that the vertical ray shooting query on \mathcal{D}_t with q' will either report the top segment of R^{par} or a bottom segment corresponding to B^{par} 's children. Note that B^{par} is the label associated with all these segments, and hence, $B_q \leftarrow B^{par}$. ◀



■ **Figure 6** (a) R' is an ancestor of R^{par} , (b) q'_1 hits the bottom segment of R' first and q'_2 hits the top segment of R^{par} first, and (c) Both q'_1 and q'_2 will not hit the bottom segment of R' first. B'' is the ancestor of B' in the tree representation.

Algorithm. Now we are ready to describe the overall algorithm. When the sweep-plane is at x -coordinate $x = x_t$, then the ray shooting data structure \mathcal{D}_t is maintained based on the boxes in \mathcal{B}_t . When the sweep-plane's next event point is the left face of a box $B \in \mathcal{B}_t$, then perform the following steps: Using Lemma 10, query \mathcal{D}_t with q' to find the parent B^{par} of B in the tree representation. Update \mathcal{D}_t by inserting the top and the bottom segments of $R \leftarrow B \cap \ell_t$. When the sweep-plane's next event point is the right face of a box $B \in \mathcal{B}_t$, then delete the top and the bottom segments of $R \leftarrow B \cap \ell_t$ from \mathcal{D}_t .

Analysis. Sorting the left and the right faces of the boxes in \mathcal{B} can be done in $O(n \log \log n)$ time [20]. At each event point, we will perform at most one vertical ray shooting query. Chan and Tsakalidis [12] present a dynamic data structure for vertical ray shooting with an expected query time of $O(\log n / \log \log n)$ and an amortized update time of $O(\log^{1/2+\epsilon} n)$. Our algorithm performs $\Theta(n)$ queries and $\Theta(n)$ updates to the ray shooting structure. Therefore, the overall expected time taken is $O(n \log n / \log \log n)$.

► **Theorem 11.** *The tree representation for n laminar axis-aligned boxes in 3D can be computed in $O(n \log n / \log \log n)$ expected time.*

► **Theorem 12.** *The tree representation for n laminar axis-aligned boxes in 2D can be computed in $O(n \log \log U)$ expected time, where the endpoints of the boxes lie on the integer grid $[U]^2$.*

Proof. We will adapt the algorithm for 3D to the 2D case. In that case, we will need a dynamic 1D point location data structure. By using the data structure of Chan [11], our algorithm can be implemented in $O(n \log \log U)$ expected time. ◀

4 Future work

As discussed in the Introduction, recently max cut on interval graphs has been shown to be NP-hard [1]. For the special case of laminar interval graphs, our works shows that max cut can be computed exactly in almost-linear time (in terms of n). However, for the other special case of max cut on unit interval graphs, the complexity status is still an open problem. Another interesting line of work is to design a polynomial time algorithm for max cut on interval graphs (and other geometric intersection graphs) with an approximation factor better than 0.878.

References

- 1 Ranendu Adhikary, Kaustav Bose, Satwik Mukherjee, and Bodhayan Roy. Complexity of maximum cut on interval graphs. In *37th International Symposium on Computational Geometry (SoCG)*, volume 189, pages 7:1–7:11, 2021.
- 2 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Advances in Discrete and Computational Geometry*, pages 1–56, 1998.
- 3 Francisco Barahona. The max-cut problem on graphs not contractible to K_5 . *Operations Research Letters*, 2(3):107–111, 1983.
- 4 Alexey Barsukov, Kaustav Bose, and Bodhayan Roy. Maximum cut on interval graphs of interval count two is np-complete. *Computing Research Repository (CoRR)*, abs/2203.06630, 2022.
- 5 Piotr Berman and Marek Karpinski. On some tighter inapproximability results (extended abstract). In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 200–209, 1999.
- 6 Hans L. Bodlaender, Celina M. H. de Figueiredo, Marisa Gutierrez, Ton Kloks, and Rolf Niedermeier. Simple max-cut for split-indifference graphs and graphs with few p_4 's. In *Experimental and Efficient Algorithms*, pages 87–99, 2004.
- 7 Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. *Nordic Journal of Computing*, 7(1):14–31, 2000.
- 8 Hans L. Bodlaender, Ton Kloks, and Rolf Niedermeier. Simple max-cut for unit interval graphs and graphs with few p_4 s. *Electronic Notes in Discrete Mathematics*, 3:19–26, 1999.
- 9 Arman Boyacı, Tinaz Ekim, and Mordechai Shalom. A polynomial-time algorithm for the maximum cardinality cut problem in proper interval graphs. *Information Processing Letters*, 121:29–33, 2017.
- 10 Arman Boyacı, Tinaz Ekim, and Mordechai Shalom. The maximum cardinality cut problem in co-bipartite chain graphs. *Journal of Combinatorial Optimization*, 35(1):250–265, 2018.
- 11 Timothy M. Chan. Persistent Predecessor Search and Orthogonal Point Location on the Word RAM. *ACM Trans. Algorithms*, 9(3), 2013.
- 12 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic Planar Orthogonal Point Location in Sublogarithmic Time. In *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99, pages 25:1–25:15, 2018.
- 13 Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending grothendieck's inequality. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 54–60, 2004.
- 14 Celina M. H. de Figueiredo, Alexsander Andrade de Melo, Fabiano de S. Oliveira, and Ana Silva. Maxcut on permutation graphs is np-complete. *Computing Research Repository (CoRR)*, abs/2202.13955, 2022.

21:12 Max Cut on Laminar Geometric Intersection Graphs

- 15 Celina MH de Figueiredo, Alexsander A de Melo, Fabiano S Oliveira, and Ana Silva. Maximum cut on interval graphs of interval count four is np-complete. *arXiv preprint*, 2020. [arXiv:2012.09804](#).
- 16 Josep Díaz and Marcin Kamiński. Max-cut and max-bisection are NP-hard on unit disk graphs. *Theoretical Computer Science*, 377(1):271–276, 2007.
- 17 M. X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.
- 18 Venkatesan Guruswami. Maximum cut on line and total graphs. *Discrete Applied Mathematics*, 92(2):217–221, 1999.
- 19 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal of Computing*, 4(3):221–225, 1975.
- 20 Yijie Han. Deterministic sorting in $o(n \log \log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004.
- 21 Klaus Jansen, Marek Karpinski, Andrzej Lingas, and Eike Seidel. Polynomial time approximation schemes for MAX-BISECTION on planar and geometric graphs. *SIAM Journal of Computing*, 35(1):110–119, 2005.
- 22 David S Johnson. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 6(3):434–451, 1985.
- 23 Satyen Kale and C. Seshadhri. Combinatorial approximation algorithms for maxcut using random walks. In *Proceedings of 2nd Symposium on Innovations in Computer Science (ICS)*, pages 367–388, 2011.
- 24 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972.
- 25 Jan Kratochvíl, Tomáš Masařík, and Jana Novotná. U-Bubble Model for Mixed Unit Interval Graphs and Its Applications: The MaxCut Problem Revisited. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170, pages 57:1–57:14, 2020.
- 26 Luca Trevisan. Max cut and the smallest eigenvalue. *SIAM Journal of Computing*, 41(6):1769–1786, 2012.

Stable Matchings with One-Sided Ties and Approximate Popularity

Telikepalli Kavitha  

Tata Institute of Fundamental Research, Mumbai, India

Abstract

We consider a matching problem in a bipartite graph $G = (A \cup B, E)$ where vertices in A rank their neighbors in a strict order of preference while vertices in B are allowed to have *weak* rankings, i.e., ties are allowed in their preferences. Stable matchings always exist in G and are easy to find, however popular matchings need not exist and it is NP-complete to decide if one exists. This motivates the “approximately popular” matching problem.

A well-known measure of approximate popularity is *low unpopularity factor*. We show that when each tie in G has length at most k , there always exists a stable matching whose unpopularity factor is at most k . Our proof is algorithmic and we compute such a stable matching in polynomial time. Our result can be considered to be a generalization of Gärdenfors’ result (1975) which showed that when rankings are strict, every stable matching is popular.

There are several applications where the size of the matching is its most important attribute. What one seeks here is a maximum matching M such that there is no maximum matching more popular than M . When rankings are weak, it is NP-hard to decide if G admits such a matching. When ties are one-sided and of length at most k , we show a polynomial time algorithm to find a maximum matching whose unpopularity factor *within* the set of maximum matchings is at most $2k$.

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

Keywords and phrases Bipartite graphs, Maximum matchings, Unpopularity factor

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.22

Funding *Telikepalli Kavitha*: Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

Acknowledgements Thanks to the reviewers for their helpful comments and suggestions.

1 Introduction

Our input is a bipartite graph $G = (A \cup B, E)$ where vertices in A are called *agents* and those in B are called *jobs*. Every vertex ranks its neighbors in an order of preference – while every agent ranks its neighbors in a strict order of preference, jobs have weak rankings, i.e., ties are allowed in their preferences. The above model is well-studied and such a model is seen when matching applicants to jobs or students to projects, e.g., the Scottish Foundation Allocation Scheme (SFAS) [21].

So in our model, every agent has a strict ordering of jobs that she finds interesting, however every job need not come up with a total order on all interested agents – here agents get grouped together in terms of their suitability to do this job, thus equally competent agents are tied together at the same rank. Our goal is to find an optimal matching in G . The classical notion of optimality is *stability*.

A matching M is stable if there is no edge that *blocks* M ; an edge (a, b) is said to block M if both a and b prefer each other to their respective assignments¹ in M . The Gale-Shapley algorithm [10] (where ties are broken arbitrarily) finds a stable matching in G in linear time.

¹ Note that being left unmatched is the least preferred option for any vertex.



© Telikepalli Kavitha;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 22; pp. 22:1–22:17



Leibniz International Proceedings in Informatics

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

Popularity. Another natural notion of optimality is *popularity*. Given any pair of matchings M and N , we say a vertex v prefers M to N if v prefers its assignment in M to its assignment in N . Let $\phi(M, N)$ be the number of vertices that prefer M to N and similarly, let $\phi(N, M)$ be the number of vertices that prefer N to M . Matching N is *more popular* than matching M if $\phi(N, M) > \phi(M, N)$.

► **Definition 1.** A matching M is *popular* if there is no matching more popular than M , i.e., $\phi(M, N) \geq \phi(N, M)$ for all matchings N .

Both stability and popularity are very desirable properties for a matching in G . While stability captures the important property that there is no pair (a, b) where both a and b are better-off by deviating from their respective assignments and pairing up with each other, popularity lays emphasis on aggregate or majority. So popularity ensures that a majority vote cannot force a migration to another matching.

When preferences are strict, stability is a stronger property and every stable matching is popular² [11]. So popular matchings always exist in G and the Gale-Shapley algorithm finds one. This is our ideal situation as we have a matching that is stable and hence, popular.

However when ties are allowed in preferences, the situation is no longer ideal – stability does not imply popularity. Even worse, popular matchings need not exist and it is NP-hard to decide if one exists [3]. Moreover, this hardness holds even in the setting of one-sided ties and every tie has length at most three [6]. Thus the popular matching problem with one-sided ties is NP-hard even under such a strict restriction on tie lengths.

Relaxing popularity. When ties are present in preferences, the lack of existence of popular matchings (and the hardness of solving the popular matching problem) motivates relaxing popularity to *near-popularity* or “low unpopularity”. A well-studied measure of unpopularity of a matching is *unpopularity factor*, introduced in [22] and studied, e.g., in [2, 8, 13, 17, 24]. Given a matching M , its unpopularity factor $u(M)$ is defined below. For any matching N ,

$$\text{let } \lambda(M, N) = \begin{cases} \phi(N, M)/\phi(M, N) & \text{if } \phi(M, N) > 0; \\ 1 & \text{if } \phi(M, N) = \phi(N, M) = 0; \\ \infty & \text{otherwise.} \end{cases}$$

Define $u(M) = \max_N \lambda(M, N)$. Thus in an election involving M and any matching N , the number of vertices that prefer N is at most $u(M)$ times the number that prefer M .

A matching M is popular if and only if $u(M) = 1$. A matching M with a low value of $u(M)$ is considered to be *near-popular*. Such a matching may lose elections but there are no heavy losses. Do near-popular matchings always exist in G ? Consider an instance $K_{n,n}$ where every agent has the same preference order $b_1 \succ b_2 \succ \dots \succ b_n$ (here b_1, \dots, b_n are the n jobs) while every job b_i has a tie of length n , i.e., it is indifferent between any two agents. It is easy to check that any matching in this instance has unpopularity factor at least $n - 1$. Thus there is *no* near-popular matching here.

However jobs in this instance have ties of length n in their preferences. Suppose no tie is very long, say every tie has length at most k (for some appropriate k). Such a restriction is quite natural in real-world applications; in fact, in many instances, we would have $k = O(1)$. Do near-popular matchings always exist then? Furthermore, are there such stable matchings?

² In an election between a stable matching S and any matching M , if vertex v prefers M to S then $M(v)$ has to prefer S to M , otherwise $(v, M(v))$ blocks S , which is forbidden. Hence $\phi(M, S) \leq \phi(S, M)$.

When preferences involve ties (even for one-sided ties of length two), a stable matching may have an unbounded unpopularity factor. Consider the following instance where $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_n\}$, and the preferences of vertices are as follows:

$$\begin{array}{ll} a_1 : & b_1 \succ b_n \\ a_i : & b_{i-1} \succ b_i \text{ for } 2 \leq i \leq n \\ b_n : & a_1 \sim a_n \\ b_i : & a_i \sim a_{i+1} \text{ for } 1 \leq i \leq n-1. \end{array}$$

So a_1 's top choice is b_1 and second choice is b_n and for $i \geq 2$, a_i 's top choice is b_{i-1} and second choice is b_i . Every $b_i \in B$ has exactly two neighbors and it is indifferent between them. The matching $M = \{(a_i, b_i) : 1 \leq i \leq n\}$ is stable. Let $N = \{(a_1, b_n)\} \cup_{i=2}^n \{(a_i, b_{i-1})\}$. Observe that $\phi(N, M) = n-1$ and $\phi(M, N) = 1$, so $u(M) \geq n-1$. It is easy to check that N is a popular matching and it is also stable.

Is there always a matching (even better, a stable matching) with low unpopularity factor in an instance with one-sided ties of bounded length? We show the following result here.

► **Theorem 2.** *Let $G = (A \cup B, E)$ be an instance with one-sided ties of length at most k . There is always a stable matching M in G such that $u(M) \leq k$. Furthermore, M can be computed in $O(k^2 \cdot mn)$ time, where $|A| = n$ and $|E| = m$.*

Observe that our earlier example of the instance $K_{n,n}$ (where each job has a tie of length n in its preference list) shows that Theorem 2 is almost tight. Recall that every matching in this instance $K_{n,n}$ has unpopularity factor at least $n-1$. Thus when jobs have ties of length k , there are instances where every matching has unpopularity factor at least $k-1$. Interestingly, we are able to show that any such instance always admits a *stable* matching with unpopularity factor at most k .

The above result generalizes the result of Gärdenfors [11] that a popular matching always exists in a bipartite graph G when rankings are strict (so $k = 1$ here). Indeed, when rankings are strict, every stable matching is popular. When ties are one-sided and each tie has length at most k , though every stable matching need not have a bounded unpopularity factor, our algorithm shows that there is always at least one stable matching M with $u(M) \leq k$. For small values of k , the *near-popularity* of M can be justified as follows: in any election involving M , if the votes in favor of M are scaled by a factor of k and we sum up these weighted votes then M never loses any election.

Maximum matchings and near-popularity. There are several applications, e.g., matching medical students to residencies [4] or allocation problems in humanitarian organizations [25, 26], where the size of the matching is more important than stability or popularity. What one seeks here is a *best* maximum matching. Let us define a “best maximum matching” as one that does not lose to another maximum matching. When rankings are strict, it is known that such a maximum matching always exists and there is an $O(mn)$ time algorithm to find one such matching [17], where $|A| = n$ and $|E| = m$.

When vertices on one side are allowed to have weak rankings, it is NP-hard to decide if such a matching exists [6]. The hardness proof in [6] showed that it is NP-hard to find a popular matching when vertices of B are allowed to have weak rankings. It is easy to check that the same proof also shows it is NP-hard to find in such an instance a maximum matching that does not lose to any maximum matching. We show the following result here.

► **Theorem 3.** *Let $G = (A \cup B, E)$ be an instance with one-sided ties of length at most k . There always exists a maximum matching M in G such that $\phi(N, M) \leq 2k \cdot \phi(M, N)$ for any maximum matching N ; furthermore, M can be computed in $O(k^2 \cdot mn^2)$ time.*

Thus we can find in polynomial time a maximum matching whose unpopularity factor *within* the set of maximum matchings is at most $2k$.

1.1 Background and related results

In the setting of strict preferences or strict rankings, popularity is a well-studied relaxation of stability – see [5] for a survey. In the setting of ties in preferences, as mentioned earlier, stability does not imply popularity; moreover, it is NP-hard to decide if a popular matching exists [3, 6], thus it is NP-hard to find a least unpopularity factor matching. Another problem that is easy for strict preferences but NP-hard for ties (even for one-sided ties) is the maximum stable matching problem which asks for a stable matching of largest size. This problem is very well-studied and several approximation algorithms in the setting of one-sided ties are known [1, 14, 15, 16, 18, 19].

Near-popularity. To the best of our knowledge, no positive results on near-popular matchings in the setting of one-sided ties are currently known. Unpopularity factor $u(\cdot)$ is a well-studied measure of unpopularity of a matching. A size-unpopularity factor trade-off in an instance $G = (A \cup B, E)$ with strict rankings is known: for any integer $k \geq 2$, there exists a matching M_k such that $u(M_k) \leq k - 1$ and $|M_k| \geq \frac{k}{k+1} \cdot |M^*|$, where M^* is a maximum matching, and such a matching M_k can be efficiently computed [17]. As shown in [17], this trade-off leads to the result that there always exists a maximum matching that is popular within the set of maximum matchings and it can be efficiently computed. In contrast to this, finding a maximum matching with the minimum number of blocking edges is NP-hard and this is NP-hard to approximate within $n^{1-\varepsilon}$, for any $\varepsilon > 0$ [4].

Matchings with low unpopularity factor in dynamic matching markets were studied in [2]. In the setting of bipartite graphs with strict rankings, when there are edge costs, it is NP-hard to find a min-cost popular matching [9]. Polynomial-time algorithms were given in [8] to find a *quasi-popular* matching M , i.e., $u(M) \leq 2$, whose cost is at most that of a min-cost popular matching. Popular matchings have also been studied in non-bipartite graphs with strict rankings. Popular matchings need not exist in such an instance G and it is NP-complete to decide if there exists one [9, 12]. It was shown in [13] that G always admits a matching with unpopularity factor $O(\log n)$ and there are non-bipartite instances with strict rankings where every matching has unpopularity factor $\Omega(\log n)$.

1.2 Our techniques

Our algorithm, roughly speaking, constructs a subgraph G' of G and returns a maximum matching in G' . The construction of G' is via a subroutine called `Propose(\cdot)`, where the argument is an agent a left unmatched in at least one maximum matching in the current G' . Such a vertex a is called *even* (formally defined in Section 2). Similar to the Gale-Shapley algorithm, in the subroutine `Propose(a)`, the agent a proposes to its neighbors in G one-by-one in decreasing order of preference.

Any vertex in B always prefers proposals from better-ranked neighbors to worse-ranked neighbors. Recall that ties are allowed in the preferences of vertices in B . So how does a job b choose between proposals of two neighbors a and a' that are tied in its ranking? The following simple rule will be key to bounding the unpopularity factor of our matching:

- Among neighbors tied in its ranking, b prefers proposals from neighbors that place b high in their preference order of neighbors in G' .

To answer our above question on a versus a' when they are tied in b 's ranking, if a (resp., a') regards b as its i -th-ranked (resp., j -th ranked) neighbor in G' , then b prefers a 's proposal if $i < j$, it prefers a' 's proposal if $j < i$, else (so $i = j$) the proposals are tied.

While $a \in A$ is even in G' and its degree is less than k , it is allowed to propose in $\text{Propose}(a)$ to its neighbors in G . When no vertex in A is even in G' , we compute a maximum matching M in the final subgraph G' and show that M is a stable matching in G with $u(M) \leq k$. This algorithm and its analysis are given in Section 3.

In order to find a desired maximum matching (as given in Theorem 3), we run the above algorithm from Section 3 in a new graph H . The graph H is a multigraph where every edge in G is replicated n times. The construction of H is inspired by the popular maximum matching algorithm from [17] where every $a \in A$ gets n chances to propose to its neighbors. Bounding the unpopularity factor of the matching computed by this algorithm within the set of maximum matchings is trickier than the analysis given in Section 3. This algorithm and its analysis are given in Section 4.

2 Preliminaries

Our algorithm will use the classical *Dulmage-Mendelsohn* decomposition [7]. Let $G = (A \cup B, E)$ be a bipartite graph and let M be a matching in G . An alternating path with respect to M is a path whose alternate edges are in M . We have $A \cup B = \mathcal{E}_M \cup \mathcal{O}_M \cup \mathcal{U}_M$, where a vertex v is in \mathcal{E}_M (resp., \mathcal{O}_M) if there is an even (resp., odd) length alternating path with respect to M from a vertex left unmatched in M to v and a vertex v is in \mathcal{U}_M if there is no alternating path from an unmatched vertex to v .

The sets \mathcal{E}_M , \mathcal{O}_M , and \mathcal{U}_M will be called the sets of *even*, *odd*, and *unreachable* vertices, respectively, with respect to M . The following theorem is well-known [20, 23].

► **Theorem 4.** *The sets \mathcal{E}_M , \mathcal{O}_M , and \mathcal{U}_M are pairwise disjoint if and only if M is a maximum matching in G . Any maximum matching in G partitions the vertex set into the same sets of even, odd, and unreachable vertices. Furthermore,*

1. *There is no edge between the sets \mathcal{E}_M and $\mathcal{E}_M \cup \mathcal{U}_M$, i.e., all the neighbors of vertices in \mathcal{E}_M are in \mathcal{O}_M .*
2. *For any maximum matching N , we have $N \subseteq (\mathcal{O}_M \times \mathcal{E}_M) \cup (\mathcal{U}_M \times \mathcal{U}_M)$. Also $|N| = |\mathcal{O}_M| + |\mathcal{U}_M|/2$, hence all vertices in $\mathcal{O}_M \cup \mathcal{U}_M$ are matched in N .*

3 Our algorithm for a near-popular matching

We present an iterative algorithm to compute a desired stable matching in an instance $G = (A \cup B, E)$ with one-sided ties, where each tie has length at most k . A relevant graph for us is $G_0 = (A \cup B_0, E_0)$ where $B_0 = B \cup \{d(a) : a \in A\}$, i.e., for each $a \in A$, we add a corresponding dummy vertex $d(a)$ called a 's *last resort job*, and $E_0 = E \cup \{(a, d(a)) : a \in A\}$. So $d(a)$ has only a as its neighbor and $d(a)$ will be the worst ranked neighbor of a .

Algorithm 1 is the overall algorithm. This algorithm constructs a subgraph G' of G_0 . The while-loop here calls the subroutine **Propose** until no agent is *even* (see Section 2) in the current G' . Recall that vertices in A or agents have strict rankings. In our algorithm, any agent a such that – (i) a is *even* in the current G' and (ii) a 's degree is less than k in the current G' – will be allowed to propose, i.e., a will be allowed to add some edges to the current G' in the subroutine $\text{Propose}(a)$.

We will show in Lemma 6 that if the set of even agents in G' is non-empty then there has to be an even agent with degree less than k in G' . Such an agent a will propose in the subroutine $\text{Propose}(a)$. When no agent is even in the subgraph G' , the construction of G' is complete. Any maximum matching in the final G' will be returned by Algorithm 1.

■ **Algorithm 1** Finding a stable matching M with $u(M) \leq k$ in $G = (A \cup B, E)$.

-
- 1: Initialize $G_0 = (A \cup B_0, E_0)$ and $G' = (A \cup B_0, \emptyset)$.
 - 2: **while** there exists an *even* agent in G' **do**
 - 3: Let a be any even agent in G' such that $\deg_{G'}(a) < k$.
 - 4: Call **Propose**(a).
 - 5: Return a maximum matching M in G' .
-

In the subroutine **Propose**(a) (see Algorithm 2), a proposes to its neighbors in G_0 one-by-one in decreasing order of preference. Each time the subroutine **Propose**(a) is called, all the proposals of a made in previous invocations of **Propose**(a) are forgotten (see line 1 of Algorithm 2) and a proposes afresh starting from its most favorite neighbor in G_0 .

■ **Algorithm 2** The subroutine **Propose**(a).

-
- 1: Delete from G' all edges incident to a . \triangleright (so all the previous proposals of a are forgotten)
 - 2: Set $i = 1$.
 - 3: **repeat**
 - 4: Let $b = a$'s most favorite neighbor in G_0 that a has not (freshly) proposed to.
 - 5: Set $\text{rank}_{G'}(a, b) = i$.
 - 6: Delete from G_0 and G' all edges between b and b 's neighbors ranked worse than a .
 \triangleright (so b will keep in G' only the best proposals that it receives in the entire algorithm)
 - 7: **if** b is isolated in G' **then**
 - 8: Add the edge (a, b) to G' .
 - 9: **else** \triangleright (a is tied with b 's neighbors in G')
 - 10: Let a' be any neighbor of b in G' .
 - 11: **if** $i < \text{rank}_{G'}(a', b)$ **then** \triangleright ($\text{rank}_{G'}(a, b) < \text{rank}_{G'}(a', b)$)
 - 12: Delete all edges incident to b in G' .
 - 13: Add the edge (a, b) to G' .
 - 14: **if** $i = \text{rank}_{G'}(a', b)$ **then** \triangleright ($\text{rank}_{G'}(a, b) = \text{rank}_{G'}(a', b)$)
 - 15: Add the edge (a, b) to G' and set $i = i + 1$.
 - 16: **until** $i = k + 1$ or a is no longer even in G' .
-

When $b \in B$ receives a proposal from a in **Propose**(a), the edge (a, b) will get added to the graph G' if one of the following two conditions is satisfied:

1. b prefers a to its current neighbors in G' – then (a, b) is unconditionally added to G' .
2. a is tied with b 's neighbors in G' and a certain value called $\text{rank}_{G'}(a, b)$ is good enough.

When a proposes to b , $\text{rank}_{G'}(a, b)$ will be b 's rank among a 's neighbors in the current G' . Let $\text{rank}_{G'}(a, b) = i$. Suppose a and a' are tied in b 's ranking, where a' is a neighbor of b in G' . When a proposes to b , i.e., when we consider the edge (a, b) in **Propose**(a):

- if $i < \text{rank}_{G'}(a', b)$ then b deletes all edges incident to it in G' and the edge (a, b) is added to G' (see line 13).
- if $i = \text{rank}_{G'}(a', b)$ then b retains all its current edges in G' and the edge (a, b) is also added to G' (see line 15). When a proposes to its next most favorite neighbor (say, b') in G_0 , it will be the case that $\text{rank}_{G'}(a, b') = i + 1$.
- if $i > \text{rank}_{G'}(a', b)$ then the edge (a, b) is not added to G' . When a proposes to its next most favorite neighbor (say, b') in G_0 , it will be the case that $\text{rank}_{G'}(a, b') = i$.

Thus among the best proposals (wrt b 's ranking) that b receives in the entire algorithm, b keeps edges in G' to only those neighbors a such that $\text{rank}_{G'}(a, b)$ is the *minimum*. In $\text{Propose}(a)$ when the edge (a, b) gets added to G' in line 8 or line 13, a will be the only neighbor of b in G' . This makes a *unreachable*³ in G' – so a is no longer *even* in G' and this will be the last iteration of the repeat-loop in this invocation of $\text{Propose}(a)$.

If a is even in G' , then there is one more iteration if $i \leq k$. Observe that i is increased when $\text{deg}_{G'}(a)$ gets increased in line 15. Thus we will always ensure that $\text{deg}_{G'}(a) \leq k$. Let us recall the other useful invariant that is maintained in our algorithm:

- all neighbors of a job b in G' are tied in b 's ranking.

► **Remark 5.** After the edge (a, b) is added to G' , the graph G' may change in later invocations of $\text{Propose}(\cdot)$. But $\text{rank}_{G'}(a, b)$ remains the same till the next invocation of $\text{Propose}(a)$.

► **Lemma 6.** *Let X be the set of even agents in G' . If $X \neq \emptyset$ then there exists $a \in X$ such that $\text{deg}_{G'}(a) < k$.*

Proof. Suppose not, i.e., suppose every $a \in X$ satisfies $\text{deg}_{G'}(a) = k$ (since $\text{deg}_{G'}(a) \leq k$). Hence there are $k \cdot |X|$ edges incident to the set X in G' . Our invariant is that all neighbors in G' of a job b are tied in b 's ranking. Since each tie has length at most k , it means that for any job b , $\text{deg}_{G'}(b) \leq k$. Hence the set of neighbors of X in G' has size at least $(k \cdot |X|)/k$, i.e., $|\text{Nbr}_{G'}(X)| \geq |X|$, where $\text{Nbr}_{G'}(X)$ is the set of neighbors of X in G' .

However it follows from the definition of “even agents” that $|X| > |\text{Nbr}_{G'}(X)|$. This is because every vertex in $\text{Nbr}_{G'}(X)$ is odd (see Theorem 4) and so for every $v \in \text{Nbr}_{G'}(X)$, there is a corresponding $M^*(v) \in X$, where M^* is any maximum matching in G' . There is also at least one vertex in X that is unmatched in M^* . Thus we get a contradiction that $|\text{Nbr}_{G'}(X)| \geq |X| > |\text{Nbr}_{G'}(X)|$. So there has to exist $a \in X$ with $\text{deg}_{G'}(a) < k$. ◀

3.1 Correctness of our algorithm

Let M be the matching returned by Algorithm 1. Delete from M all edges incident to dummy jobs, so $M \subseteq E$. Let us first show that M is a stable matching in G , i.e., no edge *blocks* M .

► **Lemma 7.** *The matching M is stable in G .*

Proof. The termination condition of Algorithm 1 is that no agent is even in G' . Hence the original matching M (before deleting edges incident to dummy jobs) is A -perfect, i.e., it matches all agents. So there is no $(a, b) \in E$ such that b prefers a to its assignment in M and a is unmatched in M , however a never got to propose to b because $\text{deg}_{G'}(a) = k$. Recall that agents add edges to G' in decreasing order of preference.

If (a, b) is in G' and a new edge (a, b') gets added to G' in the subroutine $\text{Propose}(a)$, then a prefers b to b' . Since a is *even* prior to adding the edge (a, b') , it is easy to see that after adding the edge (a, b') to G' , the agent a is either even or unreachable in G' (see footnote 3). Hence after the addition of (a, b') , the job b is odd/unreachable in G' (by Theorem 4).

Furthermore, the rank of b 's neighborhood in G' never worsens as the algorithm progresses (this is justified below). Thus in M , which is a maximum matching in G' , the job b is matched (by Theorem 4); moreover, it is matched to an agent at least as good as a .

We will next show that if an edge (a, b) gets deleted from G' then immediately after this deletion, b will be odd/unreachable in G' and b 's neighbors in G' are ranked at least as good as a . There are three cases for the deletion of an edge (a, b) from G' in our algorithm.

³ Prior to adding the new edge, a was even in G' . So all neighbors of a in the current G' will always be matched in any maximum matching, thus a has no *even* neighbor and hence a is not *odd*.

- *Case 1.* Suppose the edge (a, b) got deleted in line 1 in Algorithm 2. Then a , which is an even vertex in G' , is going to make the next round of proposals. Just prior to the deletion of (a, b) , the vertex b was *odd* in G' (since b has a neighbor a that is even). After the deletion of the edge (a, b) , the vertex b either remains odd or it becomes unreachable. Also, by our invariant, all of b 's neighbors in G' at this point are agents who are tied with a in b 's preference list.
- *Case 2.* Suppose the edge (a, b) got deleted in line 6 in Algorithm 2. This is because b received a proposal from an agent a' that b prefers to a . In this case, a' will be b 's current (and only) neighbor in G' . This makes b (and a') unreachable.
- *Case 3.* Suppose the edge (a, b) got deleted in line 12 in Algorithm 2. Just before the deletion of this edge, a was b 's neighbor in G' and all of b 's neighbors in G' were tied with a in b 's preference list and have the same $\text{rank}_{G'}$ -value as $\text{rank}_{G'}(a, b)$. As soon as b receives a proposal (say, from a') of the same rank as a and a better $\text{rank}_{G'}$ -value than $\text{rank}_{G'}(a, b)$, b deletes edges to all its current neighbors and introduces the edge (a', b) in G' . So a' will be the only neighbor of b and this makes b (and a') unreachable in G' .

So for any $(a, b) \in E$, if a prefers b to its assignment in M then b likes its partner in M at least as much as a ; hence (a, b) does not block M . Thus M is a stable matching in G . ◀

Bounding the unpopularity factor of M . For any pair of adjacent vertices v and w in G , let us define the function $\text{vote}_v(w, M)$ as follows:

$$\text{vote}_v(w, M) = \begin{cases} +1 & \text{if } v \text{ prefers } w \text{ to its assignment in } M; \\ -1 & \text{if } v \text{ prefers its assignment in } M \text{ to } w; \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\text{vote}_v(w, M)$ is v 's vote for w versus its assignment in M and this value is 0 if v is indifferent between w and $M(v)$.

Edge labels. Let us label each edge (a, b) of $E \setminus M$ by $(\text{vote}_a(b, M), \text{vote}_b(a, M))$. Note that an edge labeled $(+1, +1)$ blocks M . We showed in Lemma 7 that no edge blocks M . So every edge in $E \setminus M$ gets a label in $\{(+1, -1), (-1, +1), (-1, -1), (\pm 1, 0)\}$. We would like to show that not too many edges are labeled $(+1, 0)$.

Let $\rho = \langle \dots, e_1, f_1, e_2, f_2, \dots, e_k, f_k, \dots \rangle$ be any alternating path/cycle with respect to M , where each $e_i = (a_i, b_i)$ is in M . We will show an upper bound on the longest contiguous stretch of edges in $\rho \setminus M = \langle \dots, f_1, f_2, \dots, f_k, \dots \rangle$ that can be labeled $(+1, 0)$.

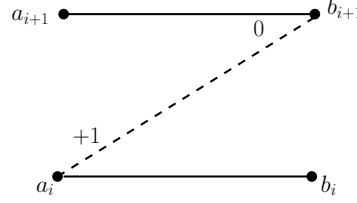
► **Lemma 8.** *If all of f_1, f_2, \dots, f_{k-1} are labeled $(+1, 0)$ then f_k cannot be labeled $(+1, 0)$.*

Proof. Suppose all of f_1, f_2, \dots, f_{k-1} are labeled $(+1, 0)$. Since $f_1 = (a_1, b_2)$ is labeled $(+1, 0)$, b_2 is indifferent between a_1 and its partner a_2 in M while a_1 prefers b_2 to its partner b_1 in M (let $b_1 = d(a_1)$ if a_1 is unmatched in M).

If the edge (a_1, b_2) is present in G' then $\text{rank}_{G'}(a_1, b_2) < \text{rank}_{G'}(a_1, b_1)$ because a_1 prefers b_2 to b_1 . Suppose the edge (a_1, b_2) is *not* present in G' . Let $\text{rank}_{G'}(a_1, b_2)$ be the $\text{rank}_{G'}$ -value with which a_1 proposed to b_2 in the very last invocation of **Propose** (a_1) . We claim $\text{rank}_{G'}(a_2, b_2) < \text{rank}_{G'}(a_1, b_2)$. This is because the edge (a_1, b_2) is missing in G' and the edge (a_2, b_2) is present in G' though b_2 is indifferent between a_1 and a_2 . Thus in both cases, we can draw the conclusion that $\text{rank}_{G'}(a_2, b_2) < \text{rank}_{G'}(a_1, b_1)$:

- in the former case, it is $\text{rank}_{G'}(a_2, b_2) = \text{rank}_{G'}(a_1, b_2) < \text{rank}_{G'}(a_1, b_1)$;
- in the latter case, it is $\text{rank}_{G'}(a_2, b_2) < \text{rank}_{G'}(a_1, b_2) \leq \text{rank}_{G'}(a_1, b_1)$.

Consider any such 3-edge sequence $\langle e_i, f_i, e_{i+1} \rangle = b_i - a_i - b_{i+1} - a_{i+1}$ in ρ (see Fig. 1) where $1 \leq i \leq k-1$. Since $f_i = (a_i, b_{i+1})$ is labeled $(+1, 0)$, the same argument as the one above for $b_1 - a_1 - b_2 - a_2$ shows that $\text{rank}_{G'}(a_{i+1}, b_{i+1}) < \text{rank}_{G'}(a_i, b_i)$. Thus we have the following chain of strict inequalities: $\text{rank}_{G'}(a_k, b_k) < \dots < \text{rank}_{G'}(a_1, b_1)$.



■ **Figure 1** The dashed edge (a_i, b_{i+1}) is labeled $(+1, 0)$ and $\text{rank}_{G'}(a_{i+1}, b_{i+1}) < \text{rank}_{G'}(a_i, b_i)$.

We have $\text{rank}_{G'}(a_1, b_1) \leq k$ since $\text{rank}_{G'}(a_1, b_1) \leq \deg_{G'}(a_1) \leq k$. So the chain of strict inequalities $\text{rank}_{G'}(a_k, b_k) < \dots < \text{rank}_{G'}(a_1, b_1)$ implies that $\text{rank}_{G'}(a_k, b_k) \leq 1$. Hence $\text{rank}_{G'}(a_k, b_k) = 1$, i.e., b_k is a_k 's most favorite neighbor in the graph G' .

We need to show that the edge $f_k = (a_k, b_{k+1})$ is *not* labeled $(+1, 0)$. If a_k prefers b_k to b_{k+1} then f_k is labeled $(-1, *)$ and we are done. The non-trivial case is when a_k prefers b_{k+1} to b_k . The following claim is proved below.

▷ **Claim 9.** If a_k prefers b_{k+1} to b_k then b_{k+1} is matched in M to an agent better than a_k .

So if a_k prefers b_{k+1} to b_k then $\text{vote}_{b_{k+1}}(a_k, M) = -1$ (by Claim 9). Hence $f_k = (a_k, b_{k+1})$ is labeled $(+1, -1)$. Thus the edge f_k is never labeled $(+1, 0)$. ◀

Proof. (of Claim 9) The agent a_k prefers b_{k+1} to b_k . Suppose the edge (a_k, b_{k+1}) is present in G_0 . Then before proposing to b_k with $\text{rank}_{G'}(a_k, b_k) = 1$, the agent a_k would have proposed to b_{k+1} with $\text{rank}_{G'}(a_k, b_{k+1}) = 1$. Since (a_k, b_{k+1}) is present in G_0 , note that a_k is as good as b_{k+1} 's neighbors in G' (wrt b_{k+1} 's ranking). Moreover, b_{k+1} would never have rejected a_k 's proposal due to *poor* $\text{rank}_{G'}$ -value as the edge (a_k, b_{k+1}) has the best possible $\text{rank}_{G'}$ -value of 1. Thus if (a_k, b_{k+1}) is in G_0 then the edge (a_k, b_{k+1}) has to be in G' and so $\text{rank}_{G'}(a_k, b_k)$ would be at least 2 since a_k prefers b_{k+1} to b_k . However $\text{rank}_{G'}(a_k, b_k) = 1$.

So the edge (a_k, b_{k+1}) is *not* present in G_0 . This means that b_{k+1} received one or more proposals from neighbors that it prefers to a_k . Hence b_{k+1} is matched in M to a neighbor that it prefers to a_k . ◀

► **Remark 10.** If ρ is an alternating cycle with respect to M then the proof of Lemma 8 shows that *all* the edges in $\rho \setminus M$ cannot be labeled $(+1, 0)$.

► **Lemma 11.** *The unpopularity factor of M is at most k .*

Proof. Let N be any matching in G . Consider $M \oplus N$ which is a set of alternating paths and alternating cycles. Let p be any alternating path in $M \oplus N$. Suppose p has even length, so $p = \langle e_1, f_1, \dots, e_t, f_t \rangle$ where for each $i \in [t]$, the edge $e_i = (a_i, b_i) \in M$ and the edge $f_i = (a_i, b_{i+1}) \in N$. We have $p \setminus M = p \cap N = \langle f_1, \dots, f_t \rangle$. Since b_{t+1} , i.e., the “job endpoint” of the last edge f_t in $p = \langle e_1, f_1, \dots, e_t, f_t \rangle$, prefers to be matched rather than be unmatched (observe that M leaves b_{t+1} unmatched) and because no edge is labeled $(+1, +1)$, the edge f_t has to be labeled $(-1, +1)$.

Suppose p has odd length, so $p = \langle e_1, f_1, \dots, e_t \rangle$. The final vertex of p (an endpoint of e_t) is an agent a_t matched in M but not in N , so a_t prefers M to N . The edge $f_{t-1} = (a_{t-1}, b_t)$ could be labeled $(+1, 0)$, however since a_t prefers M to N , the votes for N from the three

vertices a_{t-1}, b_t, a_t are $+1, 0$, and -1 , respectively. Hence this case is equivalent to the case where p has even length, i.e., the final vertex of p is in B and the last edge is labeled $(-1, +1)$. Thus in the rest of the proof, we assume without loss of generality that p has even length.

Let us extract disjoint *maximal* contiguous segments s_1, \dots, s_r from $p \cap N = \langle f_1, \dots, f_t \rangle$ such that for each i , every edge in $s_i = \langle f_{i_1}, f_{i_2}, \dots \rangle$ is labeled $(+1, 0)$ and every edge in $p \cap N$ that is labeled $(+1, 0)$ belongs to one of s_1, \dots, s_r . We know from Lemma 8 that $|s_i| \leq k - 1$ for all $i \in [r]$. The maximality of each of the segments s_1, \dots, s_r along with our observation above that f_t is labeled $(-1, +1)$ implies the following:

- for each $i \in [r]$, the edge (call it $f_{i'}$) in $p \cap N = \langle f_1, \dots, f_t \rangle$ that immediately follows the last edge in s_i has at least one “ -1 ” in its label.

For each $i \in [r]$, let us map all the edges in s_i to this edge $f_{i'}$.

For every alternating path (similarly, alternating cycle) ρ in $M \oplus N$, as done above for the alternating path p , let us perform an analogous operation of extracting such maximal contiguous segments of $(+1, 0)$ -labeled edges from $\rho \cap N$ and mapping all the edges in each such segment to the edge in $\rho \cap N$ that immediately follows this segment – this is an edge with a “ -1 ” in its label. So every edge labeled $(+1, 0)$ in N gets mapped to an edge in N that has at least one -1 in its label. Moreover, at most $k - 1$ edges are mapped to any edge.

Our goal is to bound $\phi(N, M)$ = the number of vertices that prefer N to M . Every vertex that prefers N to M contributes $+1$ to the edge of N incident to it. We know from Lemma 7 that no edge is labeled $(+1, +1)$. Thus $\phi(N, M)$ = the number of edges in $N \setminus M$ labeled by one of $(+1, 0), (+1, -1), (-1, +1)$. We have shown that the number of edges labeled $(+1, 0)$ is at most $(k - 1) \cdot$ (the number of edges labeled by one of $(+1, -1), (-1, +1), (-1, 0), (-1, -1)$). Moreover, the edge labels $(+1, -1)$ and $(-1, +1)$ have both “ $+1$ ” and “ -1 ” in them.

Hence we can conclude the following inequality on the labels of edges in $N \setminus M$: (the total number of $+1$'s in these labels) $\leq k \cdot$ (the total number of -1 's in these labels). Since $\phi(N, M)$ is the total number of $+1$'s in the labels of edges in $N \setminus M$ and $\phi(M, N)$ is at least the total number of -1 's in these edge labels, we have $\phi(N, M) \leq k \cdot \phi(M, N)$. Since this holds for any matching N , we have $u(M) \leq k$. ◀

3.2 Running time of our algorithm

We now bound the total number of times the subroutine **Propose** gets called in Algorithm 1. Let $a \in A$. After any invocation of **Propose**(a), either $\deg_{G'}(a) = k$ or a is unreachable in G' . Let $\deg(a)$ be a 's degree in G .

▷ **Claim 12.** The total number of invocations of the subroutine **Propose**(a) where $\deg_{G'}(a) = k$ at the end of this invocation is at most $k \cdot \deg(a)$.

Proof. Suppose $\deg_{G'}(a) = k$ at the end of an invocation of **Propose**(a). For **Propose**(a) to be called again, $\deg_{G'}(a)$ should be less than k . So between these two invocations of **Propose**(a), some edge (a, b) must have been deleted from G' . An edge $e = (a, b)$, once deleted from G' because b does not regard $\text{rank}_{G'}(e)$ good enough, can get re-introduced in G' in line 13 or line 15 in a later invocation of **Propose**(a). However it has to be the case that the edge e now has a smaller value of $\text{rank}_{G'}(e)$. Thus in the earlier invocation of **Propose**(a), the pair $(e, \text{rank}_{G'}(e))$ with the older value of $\text{rank}_{G'}(e)$ occurred in G' for the *last time*.

So let us charge the pair $(e, \text{rank}_{G'}(e))$ for the invocation of **Propose**(a) where $(e, \text{rank}_{G'}(e))$ occurs for the last time. Because a has at most k neighbors in G' , we have $\text{rank}_{G'}(e) \in [k]$ for any edge e in G' . Thus there are at most $k \cdot \deg(a)$ pairs $(e', \text{rank}_{G'}(e'))$ where e' is any edge incident to a . Hence the total number of invocations of **Propose**(a) where $\deg_{G'}(a) = k$ at the end of this invocation is at most $k \cdot \deg(a)$. ◀

▷ **Claim 13.** The total number of invocations of the subroutine $\text{Propose}(a)$ where a is unreachable in G' at the end of this invocation is at most $k \cdot \deg(a)$.

Proof. Suppose the vertex a is unreachable in G' at the end of an invocation of $\text{Propose}(a)$. Let (a, b) be the last edge incident to a that got added to G' in this invocation of $\text{Propose}(a)$. Observe that prior to adding the edge (a, b) to G' , the vertex a is even in G' . So it is this edge (a, b) that made a unreachable in G' . We claim that the edge (a, b) is:

1. either a *new* edge that has been added to G' for the very first time⁴
2. or it is an *old* edge but with a smaller $\text{rank}_{G'}$ -value than the previous time it was added.

We argue that either (1) or (2) stated above must have occurred because none of the old edges incident to a along with their previous $\text{rank}_{G'}$ -values was good enough for a to be unreachable in G' . This is because for the subroutine $\text{Propose}(a)$ to get invoked, the agent a has to be even in G' . Thus none of the old edges incident to a along with their previous $\text{rank}_{G'}$ -values was able to ensure a 's *unreachability* in G' . So if at the end of this invocation of $\text{Propose}(a)$, a is unreachable in G' , then either an old edge with a new $\text{rank}_{G'}$ -value or a new edge must have been introduced in G' in this invocation of $\text{Propose}(a)$.

So for every invocation of $\text{Propose}(a)$ where a is unreachable in G' at the end of this invocation, there is a pair $(e, \text{rank}_{G'}(e))$, where e is incident to a , that occurs in G' for the *first time*. We know that the number of pairs $(e, \text{rank}_{G'}(e))$ is at most $k \cdot \deg(a)$. Thus the number of invocations of $\text{Propose}(a)$ where a is unreachable in G' at the end of this invocation is at most $k \cdot \deg(a)$. Hence the claim follows. ◁

Since at the end of an invocation of $\text{Propose}(a)$, either $\deg_{G'}(a) = k$ or a is unreachable in G' , it follows that the total number of times $\text{Propose}(a)$ gets called is $O(k \cdot \deg(a))$. Thus added up over all a in A , the total number of times the subroutine Propose gets called in Algorithm 1 is $O(\sum_{a \in A} k \cdot \deg(a)) = O(k \cdot m)$.

Running time of the subroutine Propose . As done in the implementation of the Gale-Shapley algorithm, the step where a job b deletes edges to worse-ranked neighbors in G_0 is implemented in a delayed manner. That is, before an agent a proposes to any neighbor b , it first checks if the edge (a, b) is actually present in G_0 . This can be easily implemented such that the total time taken in Algorithm 1 by all the steps to delete edges from G_0 is $O(m)$.

Regarding the other steps in the subroutine Propose , we will always maintain a maximum matching M in the current G' . We will mark vertices as even/odd/unreachable – given a maximum matching in G' , this takes time linear in the size of G' . The number of edges in G' is at most $k \cdot |A| = k \cdot n$ since every $a \in A$ has degree at most k in G' .

For the subroutine $\text{Propose}(a)$ to be invoked, the agent a has to be even in G' . Hence there is an alternating path ρ between a and some agent left unmatched in M . We will update M to $M \oplus \rho$ so that a is unmatched in the resulting M . Recall that the first step of the subroutine $\text{Propose}(a)$ deletes all edges incident to a in G' ; subsequently, a proposes to its neighbors in G_0 one-by-one and adds some edges to G' .

Adding an edge between a and a neighbor that is odd/unreachable in G' maintains the property that M is a maximum matching in G' . In order to get a larger matching in G' , we need to add an edge between a and a neighbor b that is *even* in G' . The only way b can be

⁴ Note that once the edge $(a, d(a))$ gets added to G' , a remains unreachable in G' henceforth – so if a is even in G' then there is always at least one edge incident to a in G_0 that is not in G' .

even in G' is for b to be *isolated* just before the addition of the edge (a, b) to G' .⁵ So either b never received any proposal till now in Algorithm 1 or b may have had neighbors in G' , however upon receiving a 's proposal in $\text{Propose}(a)$, the vertex b had to delete all its incident edges in G' and then the edge (a, b) got added to G' .

Since b was isolated just before adding (a, b) , adding this edge makes a *unreachable* in G' (see footnote 3) and the subroutine $\text{Propose}(a)$ ends. We will update M : the edge (a, b) is added to M and if b was matched earlier, then this old edge (which is no longer in G') is deleted from M . Thus we maintain a maximum matching M in G' .

At the end of the subroutine Propose , we will search for an agent that is even in G' and with degree less than k . If such an agent is found, then the while-loop of Algorithm 1 continues; else Algorithm 1 terminates. Searching for such an agent takes time linear in the size of G' . Thus the running time of any invocation of Propose is $O(k \cdot n)$. Since there are $O(k \cdot m)$ invocations of Propose , Lemma 14 follows.

► **Lemma 14.** *The running time of Algorithm 1 is $O(k^2 \cdot mn)$.*

Theorem 2 stated in Section 1 follows from Lemma 11 and Lemma 14.

4 Maximum matchings and approximate popularity

Our goal in this section is to find a maximum matching M in $G = (A \cup B, E)$ such that $\phi(N, M) \leq 2k \cdot \phi(M, N)$ for any maximum matching N in G . Recall that G is a bipartite graph with one-sided ties of length at most k .

We will construct a new graph $H = (A \cup B, E^*)$ where every edge in G is replicated n times, recall $|A| = n$. Thus $E^* = \cup_{e \in E} \{e_1, \dots, e_n\}$, i.e., for every $e \in E$, there are n copies e_1, \dots, e_n in E^* . So H is a multigraph. Due to the presence of parallel edges in H , every vertex v in H has preferences over its incident edges (rather than its neighbors).

Let the preference order of a vertex v over its incident edges in G be $e \succeq e' \succeq e'' \succeq \dots$. In the graph H , the edges e_1, \dots, e_n (i.e., n copies of e) along with e'_1, \dots, e'_n (i.e., n copies of e') and e''_1, \dots, e''_n (i.e., n copies of e'') and so on are incident to v .

- If $v \in A$ then the preference order of v over its incident edges in H is as given below (recall that vertices in A have strict rankings in G):

$$\underbrace{e_1 \succ e'_1 \succ e''_1 \succ \dots \succ}_{\text{subscript 1 edges}} \underbrace{e_2 \succ e'_2 \succ e''_2 \succ \dots \succ}_{\text{subscript 2 edges}} \dots \underbrace{e_n \succ e'_n \succ e''_n \succ \dots}_{\text{subscript } n \text{ edges}}.$$

Vertices of A have strict rankings in H as well. For $i < j$, where $i, j \in [n]$, $v \in A$ prefers any subscript i edge incident to it to any subscript j edge incident to it. For any i , within the set of subscript i edges, v 's preference order among e_i, e'_i, e''_i, \dots is as per v 's preference order over its incident edges e, e', e'', \dots in G .

- If $v \in B$ then the preference order of v over its incident edges in H is as given below:

$$\underbrace{e_n \succeq e'_n \succeq e''_n \succeq \dots \succ}_{\text{subscript } n \text{ edges}} \underbrace{e_{n-1} \succeq e'_{n-1} \succeq e''_{n-1} \succeq \dots \succ}_{\text{subscript } n-1 \text{ edges}} \dots \underbrace{e_1 \succeq e'_1 \succeq e''_1 \succeq \dots}_{\text{subscript 1 edges}}.$$

Thus for $i < j$, where $i, j \in [n]$, $v \in B$ prefers any subscript j edge incident to it to any subscript i edge incident to it. For any i , within the set of subscript i edges, v 's preference order over e_i, e'_i, e''_i, \dots is as per v 's original preference order. So if v is indifferent between e and e' in G then v is indifferent between e_i and e'_i for every $i \in [n]$.

⁵ For an *unisolated* b to be even in G' , its neighbors have to be *odd* and no agent is ever odd in G' . Recall that as soon as the agent a becomes unreachable in $\text{Propose}(a)$, the subroutine $\text{Propose}(a)$ ends.

Thus in the graph H , while vertices in B prefer higher subscript edges, vertices in A prefer lower subscript edges. Analogous to Section 3, we have the graph H_0 where $H_0 = (A \cup B_0, E_0^*)$, $B_0 = B \cup \{d(a) : a \in A\}$, and $E_0^* = E^* \cup \{(a, d(a)) : a \in A\}$. For any $a \in A$, the edge $(a, d(a))$ is the worst ranked edge incident to a in H_0 .

The algorithm. Let us call Algorithm 1 in the graph H to construct the subgraph H' of H_0 . As before, in the subroutine **Propose**, any job b , upon receiving a proposal along an edge (say, e_i) deletes from H_0 all the edges that b ranks worse than e_i . In particular, all edges e'_j incident to b where $j < i$ get deleted from H_0 .

Algorithm 1 returns a maximum matching M^* in H' . Let M be the corresponding matching in G , i.e., M is essentially the same as M^* , however edges incident to dummy jobs are pruned from M^* and the subscripts of edges in M^* are ignored in M .

The number of edges in H is $m \cdot n$ and every tie in H is also a tie in G , so it has length at most k . Hence the running time of Algorithm 1 in H is $O(k^2 \cdot (mn)n) = O(k^2 \cdot mn^2)$. Lemma 15 and Lemma 16 prove the correctness of our algorithm. Thus Theorem 3 stated in Section 1 follows.

► **Lemma 15.** M is a maximum matching in G .

Proof. We will show there is no augmenting path with respect to M in G . Let $a_0 \in A$ be a vertex left unmatched in M . Then in the graph H' , the edge $(a_0, d(a_0))$ is in M^* . Since $(a_0, d(a_0))$ is the worst ranked edge incident to a_0 , this means that for every edge e_i , in particular, for every subscript n edge e_n incident to a_0 , either a_0 proposed along this edge e_n to the other endpoint (call it b_1) or b_1 had already deleted e_n from H_0 since it received a proposal along an edge better than e_n . Note that any edge better than e_n has to be a subscript n edge. The fact that a_0 added the edge $(a_0, d(a_0))$ to the subgraph H' implies that a_0 was *even* in H' before adding $(a_0, d(a_0))$. So at the end, every neighbor of a_0 is odd/unreachable in H' . Hence any neighbor b_1 of a_0 has to be matched along a subscript n edge in M^* .

Suppose $(a_1, b_1) \in M$. By the argument in the above paragraph, we know that it is the subscript n copy of this edge (a_1, b_1) that is present in M^* . Recall that any agent proposes or adds edges to H' in decreasing order of preference and every agent prefers lower subscript edges to higher subscript edges. So a_1 must have proposed along every subscript $(n-1)$ edge incident to it that was not yet deleted from H_0 . Furthermore, any neighbor b_2 of a_1 that deleted the subscript $(n-1)$ copy of the edge (a_1, b_2) from H_0 has to be matched along a subscript $\geq (n-1)$ edge in M^* . Thus any neighbor of a_1 has to be matched along a subscript $\geq (n-1)$ edge in M^* .

Continuing this argument, any augmenting path ρ wrt M that starts with a_0 looks as follows (the minimum possible subscripts of these edges in M^* are written below them):

$$a_0 - \underbrace{b_1 - a_1}_{\text{subscript } n} - \underbrace{b_2 - a_2}_{\text{sub. } n-1} - \underbrace{b_3 - a_3}_{\text{sub. } n-2} - \cdots - \underbrace{b_i - a_i}_{\text{sub. } n-i+1} - \cdots - \underbrace{b_n - a_n}_{\text{subscript } 1} - b.$$

Finally, the path ρ has to reach a job b that never received a proposal in H' , in other words, b is isolated in H' . So every neighbor (say, a') of b has to be matched along a subscript 1 edge since a' never proposed along the subscript 1 edge e'_1 where $e' = (a', b)$. In our augmenting path ρ , any agent matched along a subscript 1 edge has to be labeled a_t where $t \geq n$. This means ρ includes at least $n+1$ agents a_0, a_1, \dots, a_n . However this is impossible as $|A| = n$. Thus there is no augmenting path with respect to M . Equivalently, M is a maximum matching in G . ◀

22:14 Stable Matchings with One-Sided Ties and Approximate Popularity

► **Lemma 16.** *For any maximum matching N in G , $\phi(N, M) \leq 2k \cdot \phi(M, N)$.*

Before we prove Lemma 16, let us recall the following preliminaries from [17]. Let us partition $A = A_1 \cup \dots \cup A_n$ and $B = B_1 \cup \dots \cup B_n$ as follows:

- For every subscript i edge $e_i = (a, b)$ in M^* , add a to A_i and b to B_i .
- Add the vertices in A that are unmatched in M to A_n .
- Add the vertices in B that are unmatched in M to B_1 .

It follows from the definition of the sets A_i, B_i that $M \subseteq \cup_{i=1}^n (A_i \times B_i)$. It will be useful to visualize these subscripts as *levels*. So $A = A_1 \cup \dots \cup A_n$ is a partition of A into n levels where vertices in A_i are at level i and similarly, $B = B_1 \cup \dots \cup B_n$ is a partition of B into n levels where vertices in B_i are at level i . Edges in $A_i \times B_j$ where $i < j$ are said to move *upwards* and edges in $A_i \times B_j$ where $i > j$ are said to move *downwards*. Claim 17 (from [17]) shows there is no *steep* downwards edge.

▷ **Claim 17.** There is no edge between A_i and B_j where $i \geq j + 2$.

Proof. Let $a \in A_i$. We need to show a has no neighbor in B_j where $j \leq i - 2$. This argument was seen in the proof of Lemma 15. Let us consider two cases.

1. a is matched in M : Since $a \in A_i$, a proposed along all the subscript $i - 1$ edges incident to it in H_0 (that were not yet deleted) but these proposals were rejected by its neighbors. Since vertices in B prefer higher subscript edges to lower subscript edges, this means that all neighbors of a are matched along $\geq i - 1$ subscript edges. Thus a has no neighbor in B_j where $j \leq i - 2$.
2. a is not matched in M : Here $a \in A_n$ and as argued in the proof of Lemma 15, $(a, d(a))$ is in M^* and all neighbors of a are in B_n . So a has no neighbor in B_j where $j \leq n - 1$. ◁

As done in Section 3, let us label edges in $E \setminus M$ by $(\text{vote}_a(b, M), \text{vote}_b(a, M))$. Claim 18 stated below tells us that no downwards edge can have “+1” in its label.

▷ **Claim 18.** For $2 \leq j \leq n$, any edge in $A_j \times B_{j-1}$ is labeled either $(-1, -1)$ or $(-1, 0)$.

Proof. Let $e = (a, b) \in A_j \times B_{j-1}$. We need to show that e is labeled either $(-1, -1)$ or $(-1, 0)$. The job b is matched in M^* along a subscript $(j - 1)$ edge e'_{j-1} . Recall that any job prefers higher subscript edges to lower subscript edges, so if a had proposed to b along e_j then b would have accepted the proposal. Thus we can conclude that a is matched along an edge e''_j that it prefers to e_j , i.e., a prefers its partner in M to b . Thus $\text{vote}_a(b, M) = -1$.

Since agents prefer lower subscript edges to higher subscript edges, a proposed along e_{j-1} before proposing along the edge e''_j . Since b did not accept the proposal along e_{j-1} , we can conclude that b does not prefer e_{j-1} to the edge e'_{j-1} along which it is matched in M^* . Thus $\text{vote}_b(a, M) \in \{0, -1\}$. ◁

▷ **Claim 19.** Let $e \in (A_j \times B_j) \setminus M$ where $j \in [n]$. Then e cannot be labeled $(+1, +1)$.

The proof of Claim 19 is the same as the proof of Lemma 7. Note that Claims 17-19 imply that any edge labeled $(+1, +1)$ has to be an *upwards* edge.

Proof. (of Lemma 16) Let $\rho = \langle \dots, e_1, f_1, e_2, f_2, \dots \rangle$ be any alternating path/cycle with respect to M where the e -edges are in M . We will not be able to claim as done in Lemma 8 that the longest contiguous stretch of edges $\langle f_1, f_2, \dots \rangle$ labeled $(+1, 0)$ in $\rho \setminus M$ is at most $k - 1$. Now $\rho \setminus M$ can admit longer contiguous stretches of *positively labeled* edges, i.e., those labeled $(+1, 0)$ or $(+1, +1)$, by using upwards edges, i.e., edges in $\cup_{i < j} (A_i \times B_j)$, interspersed with edges in $\cup_i (A_i \times B_i)$.

Let $\rho \setminus \{\text{upwards/downwards edges}\} = \rho_1 \cup \dots \cup \rho_t$. For any $i \in [t]$, ρ_i consists of edges in $A_j \times B_j$ for some $j \in [n]$. Let $\rho_i \setminus M = \langle \dots, f_{i_1}, f_{i_2}, \dots, f_{i_{k-1}}, f_{i_k}, \dots \rangle$. We have the following claim whose proof is the same as the proof of Lemma 8.

▷ **Claim 20.** If all of $f_{i_1}, f_{i_2}, \dots, f_{i_{k-1}}$ are labeled $(+1, 0)$ then f_{i_k} cannot be labeled $(+1, 0)$.

Recall that we need to compare M only against maximum matchings. Let ρ be any *relevant* alternating path/cycle ρ wrt M , i.e., ρ occurs in $M \oplus N$ where N is a maximum matching. There are two cases here.

(1) ρ is an alternating cycle. The crucial observation is that the number of downwards edges in ρ is at least the number of upwards edges in ρ . This is because ρ is a cycle and there are no *steep* downwards edges (by Claim 17).

Recall that $\rho \setminus \{\text{upwards/downwards edges}\} = \rho_1 \cup \dots \cup \rho_t$. Each of the edges in $\rho_i \setminus M$ is in $A_j \times B_j$ for some $j \in [n]$. We know from Claim 19 that no edge in $\rho_i \setminus M$ is labeled $(+1, +1)$. Furthermore, there can be at most $k-1$ contiguous stretch of edges labeled $(+1, 0)$ in $\rho_i \setminus M$ (by Claim 20).

For any maximal contiguous segment of edges labeled $(+1, 0)$ in $\rho_i \setminus M = \langle \dots, f_{i_1}, f_{i_2}, \dots \rangle$, except possibly for the last such segment (call it ℓ_i), the maximality of each segment implies that the edge that immediately follows this segment in $\rho_i \setminus M$ has at least one “ -1 ” in its label. Analogous to Lemma 11, this “ -1 ” will *pay* for the $+1$'s in this segment.

For the last segment ℓ_i in $\rho_i \setminus M$, we have to find a “ -1 ” to pay for the $+1$'s in this segment. Note that there might be no edge in $\rho_i \setminus M$ that follows the last segment ℓ_i . This is because the edge in $\rho \setminus M$ that immediately follows ℓ_i moves to another level – so it is either a *downwards* edge or an *upwards* edge. Any downwards edge is labeled $(-1, 0)$ or $(-1, -1)$ (by Claim 18) while an upwards edge might be labeled $(+1, +1)$. Thus a downwards edge is a good edge with at least one “ -1 ” in its label while an upwards edge is possibly a bad edge with up to two $+1$'s in its label. Summarizing,

- we have to pay for the $+1$'s in the labels of upwards edges and
- we have to pay for the *last-segments* ℓ_1, \dots, ℓ_t in ρ_1, \dots, ρ_t , respectively.

We have t last-segments and some $s \leq t$ upwards edges. Our goal is to show that there are enough downwards edges in ρ so that $2k$ times their number is an upper bound on the total number of $+1$'s in the upwards edges and in the last-segments.

Consider the following list: $\ell_1, f'_1, \ell_2, f'_2, \dots, \ell_t, f'_t$, where ℓ_1, \dots, ℓ_t are all the last-segments and for $i \in [t]$, f'_i is the upwards/downwards edge in $\rho \setminus M$ that immediately follows $\rho_i \setminus M$. Each edge in ℓ_i is labeled $(+1, 0)$ while the edge f'_i (if it is an upwards edge) could be labeled $(+1, +1)$.

Let $F = \{f'_1, \dots, f'_t\}$ and let $X \subseteq F$ be the set of downwards edges and let $Y \subseteq F$ be the set of upwards edges. We know that the number of downwards edges is at least the number of upwards edges, i.e., $|X| \geq |Y| = s$ and $|X| + |Y| = t$. So $t - s \geq s$, i.e., $s \leq t/2$. For convenience, let us assume that $X = \{f'_1, \dots, f'_{t-s}\}$ and $Y = \{f'_{t-s+1}, \dots, f'_t\}$.

Our charging mechanism is as follows:

- For each $1 \leq i \leq t - s$, the “ -1 ” in edge f'_i will pay for all the $+1$'s in the segment ℓ_i .
- For each $1 \leq i \leq s$, the “ -1 ” in f'_i will also pay for all the $+1$'s in the segment ℓ_{t-s+i} and moreover, for the $+1$'s in the edge f'_{t-s+i} .

Since there are at most $(k-1)$ $+1$'s in any ℓ_j and at most two $+1$'s in any upwards edge, it follows that the total number of $+1$'s that any “ -1 ” pays for is at most $2(k-1) + 2 = 2k$.

(2) ρ is an even length alternating path. So ρ has one endpoint that is unmatched in M . There are two subcases here based on whether the unmatched endpoint is in A or in B .

- Suppose the unmatched endpoint of ρ is in A . Recall that any vertex $a \in A$ that is unmatched in M is in the set A_n . Since ρ has even length, the final vertex in ρ is a matched agent $a' \in A_i$ for some $i \leq n$. Because there are no steep downwards edges, the number of downwards edges in ρ is at least the number of upwards edges.

Thus the following crucial property – the number of downwards edges in ρ is at least the number of upwards edges – used in case (1) holds in this subcase as well. Now the rest of the argument that the total number of +1's in ρ is at most $2k \cdot$ (the number of -1 's in ρ) is the same as given in case (1).

- Suppose the unmatched endpoint of ρ is in B . Recall that any vertex $b \in B$ that is unmatched in M is in the set B_1 . Observe that any neighbor a of an unmatched $b \in B$ is in A_1 . This is because a never proposed along the subscript 1 edge corresponding to (a, b) and this means that a is matched along a subscript 1 edge.

The other endpoint of ρ is a vertex $b' \in B$ that is matched in M . Thus the final vertex b' is in B_i for some $i \geq 1$ and its partner in M (call it a') is in A_i . Hence the alternating path ρ traverses from $a' \in A_i$ where $i \geq 1$ to $a \in A_1$. Since there are no steep downwards edges, the crucial property that the number of downwards edges in ρ is at least the number of upwards edges holds here as well. The rest of the argument that the total number of +1's in ρ is at most $2k \cdot$ (the number of -1 's in ρ) is the same as in case (1).

For any maximum matching N , the symmetric difference $M \oplus N$ is a collection of alternating cycles and alternating paths of even length. For any such alternating path or alternating cycle ρ , we know that (the number of +1's in ρ) $\leq 2k \cdot$ (the number of -1 's in ρ). Equivalently, $\phi(M \oplus \rho, M) \leq 2k \cdot \phi(M, M \oplus \rho)$. Hence $\phi(N, M) \leq 2k \cdot \phi(M, N)$. ◀

5 Conclusions and open problems

We showed that any bipartite graph $G = (A \cup B, E)$ where vertices in A have strict rankings over their neighbors and vertices in B have weak rankings with ties of length at most k admits a stable matching M with unpopularity factor at most k . So no matching N can win more than $k/(k+1)$ -fraction of votes in a head-to-head election against M , where vertices are voters. We showed a polynomial time algorithm to find such a matching M . It is easy to show instances with one-sided ties of length at most k where every matching has unpopularity factor at least $k-1$. Thus our bound on unpopularity factor is almost tight.

An open problem is to extend the results shown here to *two-sided* ties of bounded length, i.e., given an instance $G = (A \cup B, E)$ where all vertices are allowed to have weak rankings with ties of length at most k , is there always a matching in G with unpopularity factor at most k ? Furthermore, is there always such a stable matching? Is it easy to find one?

References

- 1 F. Bauckholt, K. Pashkovich, and L. Sanitá. On the approximability of the stable marriage problem with one-sided ties. [arXiv:1805.05391](https://arxiv.org/abs/1805.05391).
- 2 S. Bhattacharya, M. Hofer, C.-C. Huang, T. Kavitha, and L. Wagner. Maintaining near-popular matchings. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)(II)*, pages 504–515, 2015.
- 3 P. Biro, R. W. Irving, and D. F. Manlove. Popular matchings in the marriage and roommates problems. In *Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC)*, pages 97–108, 2010.

- 4 P. Biro, D. F. Manlove, and S. Mittal. Size versus stability in the marriage problem. *Theoretical Computer Science*, 411:1828–1841, 2010.
- 5 Á. Cseh. Popular matchings. Trends in Computational Social Choice, Ulle Endriss (ed.), 2017.
- 6 Á. Cseh, C.-C. Huang, and T. Kavitha. Popular matchings with two-sided preferences and one-sided ties. *SIAM Journal on Discrete Mathematics*, 31(4):2348–2377, 2017.
- 7 A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.
- 8 Y. Faenza and T. Kavitha. Quasi-popular matchings, optimality, and extended formulations. *Mathematics of Operations Research*, 47(1):427–457, 2022.
- 9 Y. Faenza, T. Kavitha, V. Powers, and X. Zhang. Popular matchings and limits to tractability. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2790–2809, 2019.
- 10 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.
- 11 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20:166–173, 1975.
- 12 S. Gupta, P. Misra, S. Saurabh, and M. Zehavi. Popular matching in roommates setting is NP-hard. *ACM Transactions on Computation Theory*, 13(2):1–20, 2021.
- 13 C.-C. Huang and T. Kavitha. Near-popular matchings in the roommates problem. *SIAM Journal on Discrete Mathematics*, 27(1):43–62, 2013.
- 14 C.-C. Huang and T. Kavitha. Improved approximation algorithms for two variants of the stable marriage problem with ties. *Mathematical Programming*, 154(1):353–380, 2015.
- 15 K. Iwama, S. Miyazaki, and N. Yamauchi. A 1.875-approximation algorithm for the stable marriage problem. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 288–297, 2007.
- 16 K. Iwama, S. Miyazaki, and N. Yamauchi. A 25/17-approximation algorithm for the stable marriage problem with one-sided ties. *Algorithmica*, 68(3):758–775, 2014.
- 17 T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.
- 18 Z. Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60(1):3–20, 2011.
- 19 C.-K. Lam and C. G. Plaxton. A $(1 + 1/e)$ -approximation algorithm for maximum stable matching with one-sided ties and incomplete lists. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2823–2840, 2019.
- 20 L. Losász and M. D. Plummer. *Matching theory*. North-Holland, Mathematics Studies 121, 1986.
- 21 D. F. Manlove and R. W. Irving. Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. *Journal of Combinatorial Optimization*, 16:279–292, 2008.
- 22 M. McCutchen. The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 593–604, 2008.
- 23 W. R. Pulleyblank. Chapter 3, matchings and extensions. The Handbook of Combinatorics, R.L. Graham, M. Grötschel, and L. Lovasz (ed.), 1995.
- 24 S. Ruangwises and T. Itoh. Unpopularity factor in the marriage and roommates problems. *Theory of Computing Systems*, 65(3):579–592, 2021.
- 25 M. Soldner. Optimization and measurement in humanitarian operations: Addressing practical needs. PhD thesis, Georgia Institute of Technology, 2014.
- 26 A.C. Trapp, A. Teytelboym, A. Martinello, T. Andersson, and N. Ahani. Placement optimization in refugee resettlement. Working paper, 2018.

Geometry Meets Vectors: Approximation Algorithms for Multidimensional Packing

Arindam Khan  

Department of Computer Science and Automation,
Indian Institute of Science, Bengaluru, India

Eklavya Sharma  

Department of Industrial & Enterprise Systems Engineering,
University of Illinois at Urbana-Champaign, IL, USA

K. V. N. Sreenivas 

Department of Computer Science and Automation,
Indian Institute of Science, Bengaluru, India

Abstract

We study the generalized multidimensional bin packing problem (GVBP) that generalizes both geometric packing and vector packing. Here, we are given n rectangular items where the i^{th} item has width $w(i)$, height $h(i)$, and d nonnegative weights $v_1(i), v_2(i), \dots, v_d(i)$. Our goal is to get an axis-parallel non-overlapping packing of the items into square bins so that for all $j \in [d]$, the sum of the j^{th} weight of items in each bin is at most 1. This is a natural problem arising in logistics, resource allocation, and scheduling. Despite being well-studied in practice, approximation algorithms for this problem have rarely been explored.

We first obtain two simple algorithms for GVBP having asymptotic approximation ratios $6(d+1)$ and $3(1 + \ln(d+1) + \varepsilon)$. We then extend the Round-and-Approx (R&A) framework [3, 6] to wider classes of algorithms, and show how it can be adapted to GVBP. Using more sophisticated techniques, we obtain better approximation algorithms for GVBP, and we get further improvement by combining them with the R&A framework. This gives us an asymptotic approximation ratio of $2(1 + \ln((d+4)/2)) + \varepsilon$ for GVBP, which improves to $2.919 + \varepsilon$ for the special case of $d = 1$. We obtain further improvement when the items are allowed to be rotated. We also present algorithms for a generalization of GVBP where the items are high dimensional cuboids.

2012 ACM Subject Classification Theory of computation \rightarrow Packing and covering problems

Keywords and phrases Bin packing, rectangle packing, multidimensional packing, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.23

Related Version *arXiv Version*: <https://arxiv.org/abs/2106.13951>

Funding *Arindam Khan*: Research partly supported by Pratiksha Trust Young Investigator Award, Google India Research Award, and Google ExploreCS Award.

Acknowledgements We thank Nikhil Bansal, Thomas Rothvoss, and anonymous reviewers for their helpful comments.

1 Introduction

Bin packing and knapsack problems are classical NP-hard optimization problems. Two classical generalizations of these problems: geometric packing and vector packing have been well-studied from the 1980s [14, 17]. Geometric packing considers the packing of rectangular items, whereas, in vector packing items are multidimensional vectors. However, often in practice, we encounter a mixture of geometric and vector constraints. Consider the following airlines cargo problem [42]: We have boxes to load in an airline cargo container. In addition to the geometric constraint that all the boxes must fit within the container, we also have a constraint that the total weight of the loaded boxes is within a specified capacity. Thus, in this problem, three dimensions are geometric and the weight is a vector constraint.



© Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 23; pp. 23:1–23:22



Leibniz International Proceedings in Informatics

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

Weight has been an important constraint to consider for packing in logistics and supply chain management, e.g., cranes and other equipment can be damaged by the bins being too heavy [1]. When different cargoes are packed into a fleet of aircraft for transport, one needs the individual cargoes to be not too heavy to ensure stability and less fuel consumption [2]. Similar problems find applications in vehicle routing with loading constraints [8]. Many practical heuristics [48, 50] have been proposed for such problems. Many companies (such as Driw, Boxify, Freightcom) and practical packages [53] have considered the problem. Often, we also want to limit other attributes, like the amount of magnetism, radioactivity, or toxicity. Each such property can be considered an additional vector dimension.

Such multidimensional packing problems also get attention due to their connections with fair resource allocation [43]. In recent years, a considerable amount of research has focused on group fairness [31, 51] such that the algorithms are not biased towards (or against) some groups or categories. One such notion of fairness is *restricted dominance* [7], which upper bounds the number (or size) of items from a category. These different categories can be considered as dimensions. E.g., in a container packing problem for flood relief, one needs to ensure that the money spent on a container is fairly distributed among different types of items (such as medicine, food, garments). Hence, for each category, there is an upper bound on the value that can go into a container.

Formally, we are given n items $I := \{1, 2, \dots, n\}$ that are (d_g, d_v) -dimensional, i.e., item i is a d_g -dimensional cuboid of lengths $\ell_1(i), \ell_2(i), \dots, \ell_{d_g}(i)$ and has d_v non-negative weights $v_1(i), v_2(i), \dots, v_{d_v}(i)$. A (d_g, d_v) -dimensional bin is a d_g -dimensional cuboid of length 1 in each geometric dimension and weight capacity 1 in each of the d_v vector dimensions. A feasible packing of items into a bin is a packing where items are packed parallel to the axes without overlapping, and for all $j \in [d_v]$, the sum of the j^{th} vector dimension of the items in the bin is at most 1 (see Definition 14 in Appendix A for a more formal definition of (d_g, d_v) packing). In the (d_g, d_v) bin packing problem (BP), we have to feasibly pack all items into the minimum number of bins. In the (d_g, d_v) knapsack problem (KS), each item i also has an associated nonnegative profit $p(i)$, and we have to feasibly pack a maximum-profit subset of the items into a single bin (also called “knapsack”). (d_g, d_v) packing problems generalize both d_g -dimensional geometric packing (when $d_v = 0$) and d_v -dimensional vector packing (when $d_g = 0$). Already for vector bin packing, if d_v is part of the input, there is an approximation hardness of $d_v^{1-\varepsilon}$, unless $\text{NP}=\text{ZPP}$ [5]. Thus, throughout the paper we assume both d_g and d_v to be constants.

1.1 Our Results

We study the first approximation algorithms for general (d_g, d_v) BP, with a focus on $d_g = 2$. We give two simple algorithms for $(2, d)$ BP, called `simplePack` and `betterSimplePack`, having asymptotic approximation ratios (AARs) of $6(d + 1)$ and $3(1 + \ln(d + 1)) + \varepsilon$, respectively, for any $\varepsilon > 0$. Getting an AAR better than $O(\log d)$ for d -D VBP is NP-hard [46], so `betterSimplePack`’s AAR as a function of d is tight up to a constant factor. For $d = 1$, `betterSimplePack`’s AAR improves to $\approx 4.216 + \varepsilon$.

Next, we modify the Round-and-Approx (R&A) framework [6] so that it works for (d_g, d_v) BP. We combine R&A with the `simplePack` algorithm to get an AAR of $2(1 + \ln(3(d + 1))) + \varepsilon$ for $(2, d)$ BP. This improves upon the AAR of `betterSimplePack` for $d \geq 3$.

In Section 5, we obtain a more sophisticated algorithm for $(2, d)$ BP, called `cbPack`, that fits into the R&A framework and has an even better AAR. Table 1 lists the AARs of all our algorithms for $(2, d)$.

■ **Table 1** Asymptotic approximation ratios of our algorithms for $(2, d)$ BP.

Algorithm	AAR for $(2, d)$ BP	AAR for $(2, 1)$ BP
<code>simplePack</code>	$6(d + 1)$	12
<code>betterSimplePack</code>	$3(1 + \ln(d + 1)) + \varepsilon$	$3(1 + \ln(\frac{3}{2})) + \varepsilon \approx 4.216 + \varepsilon$
<code>simplePack</code> with R&A	$2(1 + \ln(3(d + 1))) + \varepsilon$	$2(1 + \ln 6) + \varepsilon \approx 5.5835 + \varepsilon$
<code>cbPack</code> with R&A (without rotation)	$2(1 + \ln(\frac{d+4}{2})) + \varepsilon$	$2(1 + \ln(\frac{19}{12})) + \varepsilon \approx 2.919 + \varepsilon$
<code>cbPack</code> with R&A (with rotation)	$2(1 + \ln(\frac{d+3}{2})) + \varepsilon$	$2(1 + \ln(\frac{3}{2})) + \varepsilon \approx 2.811 + \varepsilon$

We also show how to extend `simplePack` and `betterSimplePack` to (d_g, d_v) BP to obtain AARs $2b(d_v + 1)$ and $b(1 + \ln(d_v + 1) + \varepsilon)$, respectively, where $b := 9$ when $d_g = 3$, and $b := 4^{d_g} + 2^{d_g}$ when $d_g > 3$. We also give a similar algorithm for (d_g, d_v) KS, having an approximation ratio $b(1 + \varepsilon)$.

1.2 Related Work

The bin packing problem (BP) has been the cornerstone of approximation algorithms [29]. The standard performance measure for BP algorithms is the asymptotic approximation ratio (AAR). An asymptotic polynomial time approximation scheme (APTAS) for BP was given by Fernandez de la Vega and Lueker [17], using linear grouping. Note that 1-D BP can be considered as $(1, 0)$ BP as well as $(0, 1)$ BP. The present best approximation algorithm for 1-D BP returns a packing in at most $\text{opt} + O(\log \text{opt})$ bins, where opt is the optimal number of bins [25]. Knapsack problem (KS) is one of Karp’s 21 NP-complete problems. Lawler gave an FPTAS [40] for KS. For surveys on BP and KS, see [13, 33].

In [17], a $(d + \varepsilon)$ -asymptotic approximation algorithm was given for the d -dimensional vector bin packing (d -D VBP). Chekuri and Khanna gave a $\ln d + O(1)$ approximation for d -D VBP [11]. The study of 2-D geometric bin packing (2-D GBP) was initiated by [14]. Caprara gave a T_∞^{d-1} -asymptotic approximation Harmonic-Decreasing-Height (HDH) algorithm for d -D GBP [9], where $T_\infty \approx 1.6901$. This is still the best known approximation for d -D GBP for $d \geq 3$. Both 2-D GBP and 2-D VBP do not admit an APTAS [4, 52, 45]. For large d , getting better than $O(\log d)$ asymptotic approximation for d -D VBP is NP-hard [46].

Bansal, Caprara, and Sviridenko [3] introduced the *Round-and-Approx (R&A)* framework to obtain improved approximations for both 2-D GBP and d -D VBP. The R&A framework is a two stage process. First, a (possibly exponential-sized) set covering LP relaxation (called configuration LP) is solved approximately. Then, a randomized rounding procedure is applied for a few steps to pack a subset of items, after which only a “small” fraction of items (called the *residual instance*) remain unpacked. In the second step, the residual instance is packed using a *subset-oblivious* algorithm. Intuitively, given a *random* subset S of I where each element occurs with probability about $1/k$, a ρ -approximate subset-oblivious algorithm produces a packing of S in approximately $\rho \text{opt}(I)/k$ bins. In the R&A framework, one can obtain a $(1 + \ln \rho)$ -approximation algorithm using a ρ -approximate subset oblivious algorithm. Two algorithms, 1-D BP APTAS [17] and HDH [9] were shown to be subset-oblivious based on various properties of dual-weighting functions. This led to an AAR of $(1 + \ln(1.69))$ and $(1 + \ln d)$ for 2-D GBP and d -D VBP, respectively. However, it was cumbersome to extend subset-obliviousness to wider classes of algorithms.

Bansal and Khan [6] later extended the R&A framework for 2-D GBP to *rounding-based* algorithms, where the large dimensions are rounded up to $O(1)$ values and the packing of items is container-based, i.e., each bin contains a constant number of rectangular regions

called containers and items are packed into containers. For 2-D GBP, they used an algorithm with an AAR of 1.5 [27, 44] to obtain the present best AAR of $(1 + \ln 1.5) \approx 1.405$. For d -D VBP, Bansal et al. [5] used the R&A framework combined with a multi-objective budgeted matching problem, to obtain the present best AAR of $(0.81 + o_d(1) + \ln d)$.

Multidimensional knapsack is also well-studied. For d -D vector knapsack (d -D VKS), Frieze and Clarke gave a PTAS [18]. For 2-D geometric knapsack (GKS), Jansen and Zhang [28] gave a $(2 + \varepsilon)$ -approximation algorithm, while the present best approximation ratio is $\frac{17}{9} + \varepsilon$ [20]. It is not even known whether 2-D GKS is APX-hard or not. There are many other related important geometric packing problems, such as strip packing [21, 19] and maximum independent set of rectangles [36, 22, 23]. For surveys on multidimensional packing, see [12, 35].

1.3 Technical Contribution

One of our main contributions is the enhancement of R&A framework [6] to wider applications.

First, R&A framework now also works with (d_g, d_v) -dimensional items, unifying the approach for geometric and vector packing. To use R&A, we need to solve the configuration LP of the corresponding bin packing problem. All previous applications (d -D VBP and 2-D GBP) of R&A solved the configuration LP within $(1 + \varepsilon)$ factor using a $(1 + O(\varepsilon))$ -approximate solution to (a variant of) KS. Due to the unavailability of a PTAS for (a variant of) $(2, d)$ KS, we had to use a different linear programming algorithm [47] that uses an η -approximation algorithm for KS to $(1 + \varepsilon)\eta$ -approximately solve the configuration LP of the corresponding BP problem, for any constants $1 < \eta, 0 < \varepsilon < 1$.

Second, we introduce more freedom in choosing the packing structure. Unlike the R&A framework in [6] that worked only for container-based packing, we allow either relaxing the packing structure to non-container-based (like in `simplePack`) or imposing packing constraints in addition to being container-based (like in `cbPack`). This generalization can help in obtaining improved algorithms for other problems related to bin packing.

Finally, we allow rounding items in ways other than rounding up, if we can find a suitable way of *unrounding* a packing of rounded items. In `cbPack`, we round down the width and height of some items to 0, and in `simplePack`, we round each $(2, d)$ -dimensional item i to an item of width 1, height x and each vector dimension x , where x is a value depending on the original dimensions of i . As shown in [35], if the large coordinates of items are rounded up to $O(1)$ types, we cannot get an AAR better than d and $4/3$ for d -D VBP and 2-D GBP, respectively. However, as we now allow rounding down, the R&A framework may now work with algorithms having better AARs.

We also fix a minor error in the R&A framework of [35]. See Appendix C.1 for details.

In [3], it was mentioned: “One obstacle against the use of R&A for other problems is the difficulty in deriving subset-oblivious algorithms (or proving that existing algorithms are subset oblivious).” We expect that our progress will help in understanding the power of R&A to extend it to other set-cover type problems, e.g. round-SAP [32] and round-UFP [16, 32].

Our another major contribution is handling of the $(2, d)$ BP problem. This problem presents additional challenges over pure geometric BP, and our algorithm `cbPack` demonstrates how to circumvent them. For example, in geometric packing, items of low total area can be packed into a small number of bins using the NFDH algorithm [14]. This need not be true when items have weights, since the geometric dimensions can be small but the vector dimensions may be large. To handle this, we divide the items into different classes based on density (i.e., weight/area). We use the facts that items of low density and low total area can be packed into a small number of bins, and items of high density that fit into a bin have low

total area. Also, in geometric packing, we can sometimes move items with similar geometric dimensions across different bins (like in *linear grouping* [17, 44]). Vector dimensions again create problems here. To handle this, we only move items of similar geometric dimensions and density. This leads to a more *structured* packing and we show how to find such a near-optimal structured packing efficiently. Due to space limitations, we defer the description and analysis of `cbPack` to Section 5 and the full version's Appendix F [38].

2 Preliminaries and Notation

Let \mathcal{I} be the set of all valid inputs to a minimization problem \mathcal{P} . For any input $I \in \mathcal{I}$, let $\text{opt}(I)$ be the cost of the optimal solution and $|\mathcal{A}(I)|$ be the cost of algorithm \mathcal{A} 's output on I . Define the *approximation ratio* $\rho_{\mathcal{A}}$ and the *asymptotic approximation ratio* (AAR) $\rho_{\mathcal{A}}^{\infty}$ as $\rho_{\mathcal{A}} := \sup_{I \in \mathcal{I}} \{|\mathcal{A}(I)|/\text{opt}(I)\}$, and $\rho_{\mathcal{A}}^{\infty} := \limsup_{z \rightarrow \infty} \sup_{I \in \mathcal{I}} \left\{ |\mathcal{A}(I)|/\text{opt}(I) \mid \text{opt}(I) = z \right\}$, respectively. Intuitively, AAR is \mathcal{A} 's performance for inputs with large opt .

Let $[n] := \{1, 2, \dots, n\}$. Let $\text{poly}(n)$ be the set of polynomial and sub-polynomial functions of n . Define v_{\max} , vol , and span as follows: $v_{\max}(i) := \max_{j=1}^{d_v} v_j(i)$, $\text{vol}(i) := \prod_{j=1}^{d_g} \ell_j(i)$, $\text{span}(i) := \max(\text{vol}(i), v_{\max}(i))$. $\text{span}(i)$ is, intuitively, the measure of *largeness* of item $i \in [n]$. For convenience, let $v_0(i) := \text{vol}(i)$. Assume w.l.o.g. that $\text{vol}(i) = 0$ implies $(\forall j \in [d_g], \ell_j(i) = 0)$. For a set I of items, given a function $f : I \mapsto \mathbb{R}$, for $S \subseteq I$, define $f(S) := \sum_{i \in S} f(i)$. This means, e.g., $\text{vol}(S) := \sum_{i \in S} \text{vol}(i)$. For any bin packing algorithm \mathcal{A} , let $\mathcal{A}(I)$ be the resulting bin packing of items I , and let $|\mathcal{A}(I)|$ be the number of bins in $\mathcal{A}(I)$. Define $\text{opt}(I)$ as the minimum number of bins needed to pack I . The following lemma relates span with opt .

► **Lemma 1.** For (d_g, d_v) items I , $\lceil \text{span}(I) \rceil \leq (d_v + 1) \text{opt}(I)$.

Proof. Let $m = \text{opt}(I)$. In an optimal packing, let J_j be the items in the j^{th} bin. Then $\lceil \text{span}(I) \rceil = \left\lceil \sum_{k=1}^m \sum_{i \in J_k} \max_{j=0}^{d_v} v_j(i) \right\rceil \leq \left\lceil \sum_{k=1}^m \sum_{j=0}^{d_v} v_j(J_k) \right\rceil \leq (d_v + 1)m$. ◀

For $d_g = 2$, let $w(i) := \ell_1(i)$, $h(i) := \ell_2(i)$ be the width and height of item i , respectively. The area of item i is $a(i) := w(i)h(i) = \text{vol}(i)$. The items in $(2, 0)$ BP are called “rectangles”.

2.1 Configuration LP

For a (d_g, d_v) bin packing instance I containing n items, a configuration of I is a packing of a subset of items of I into a bin. Let \mathcal{C} be the set of all configurations of I . The *configuration matrix* of I is a matrix $A \in \{0, 1\}^{n \times |\mathcal{C}|}$ where $A[i, C]$ is 1 if configuration C contains item i and 0 otherwise. To solve the bin packing problem, it is sufficient to decide the number of bins of each configuration. This gives us the following linear programming relaxation, called a configuration LP:

$$\min_{x \in \mathbb{R}^{|\mathcal{C}|}} \sum_{C \in \mathcal{C}} x_C \quad \text{where} \quad Ax \geq \mathbf{1} \text{ and } x \geq 0.$$

Even though $|\mathcal{C}|$ can be exponential in n , any feasible solution x to the configuration LP may have a polynomial-sized representation if the size of $\text{support}(x)$ is polynomially bounded in n .

3 Simple Algorithms

In this section, we look at simple algorithms for $(2, d)$ BP. They are based on the following simple corollary of Steinberg's algorithm [49].

► **Lemma 2** (Section 3 in [28]). *Let I be a set of rectangles where $a(I) \leq 1$. Then I can be packed into 3 bins in $O(n \log^2 n / \log \log n)$ time.*

Let I be a $(2, d)$ BP instance. Let $\hat{I} := \{\text{span}(i) : i \in I\}$, i.e., \hat{I} is a 1-D BP instance. The algorithm `simplePack`(I) first runs the Next-Fit algorithm [30] on \hat{I} . Let $[\hat{J}_1, \hat{J}_2, \dots, \hat{J}_m]$ be the resulting bin packing of \hat{I} into m bins. For each $\hat{J}_j \subseteq \hat{I}$, let J_j be the corresponding items from I . Then $\forall k \in [d_v]$, $v_k(J_j) \leq 1$ and $\text{vol}(J_j) \leq 1$. `simplePack` then uses the algorithm of Lemma 2 to pack each J_j into at most 3 bins, giving a packing of I into at most $3m$ bins. By the property of Next-Fit [30], we get that $m \leq \lceil 2 \text{size}(\hat{I}) \rceil = \lceil 2 \text{span}(I) \rceil$. By Lemma 1, we get $3 \lceil 2 \text{span}(I) \rceil \leq 6(d+1) \text{opt}(I)$. This gives us the following theorem.

► **Theorem 3.** *For $(2, d)$ BP, `simplePack` uses at most $3 \lceil 2 \text{span}(I) \rceil$ bins, so it is a $6(d+1)$ -approximation algorithm. It runs in $O(nd + n \log^2 n / \log \log n)$ time.*

The algorithm `betterSimplePack` first computes \tilde{I} , which is a $(d+1)$ -D VBP instance obtained by replacing the geometric dimensions of each item $i \in I$ by a single vector dimension $a(i)$. It computes a bin packing of \tilde{I} using any algorithm \mathcal{A} . It then uses the algorithm of Lemma 2 to pack I into at most $3|\mathcal{A}(\tilde{I})|$ bins.

Note that $\text{opt}(\tilde{I}) \leq \text{opt}(I)$. If \mathcal{A} has AAR α , then $|\mathcal{A}(\tilde{I})| \leq \alpha \text{opt}(\tilde{I}) + O(1)$. Therefore, `betterSimplePack` has AAR 3α . The $(d+1)$ -D VBP algorithm by [3] (parametrized by a constant $\varepsilon > 0$) gives $\alpha = 1 + \ln(d+1) + \varepsilon$ and the algorithm by [5] gives $\alpha = 1.5 + \ln((d+2)/2) + \varepsilon$ (improves to $\alpha = 1 + \ln(1.5) + \varepsilon$ for $d = 1$).

Similarly, we can get a $3(1 + \varepsilon)$ -approximation algorithm for $(2, d)$ KS (see Appendix D of the full version [38]).

Although `simplePack`'s AAR is worse than `betterSimplePack`, `simplePack`'s output is upper-bounded in terms of span, which is a useful property. Hence, we will use it as a subroutine in other algorithms (like `cbPack`).

The algorithms for $(2, d)$ packing can be extended to (d_g, d_v) packing. We just need an algorithm for the following problem: *given a set J of d_g -dimensional cuboids where $\text{vol}(J) \leq 1$, pack J into a small number of bins.* We used Lemma 2 when $d_g = 2$. When $d_g = 3$, we can use the algorithm of Diedrich et al. (Section 2 of [15]) to pack J into at most 9 bins. For $d_g > 3$, we can pack J into at most $4^{d_g} + 2^{d_g}$ bins using a variant of the HDH₄ algorithm [10] (see Appendix C of the full version [38]). Hence, `simplePack` will use $b \lceil 2 \text{span}(I) \rceil$ bins, where $b := 3$ when $d_g = 2$, $b := 9$ when $d_g = 3$, and $b := 4^{d_g} + 2^{d_g}$ when $d_g > 3$. Therefore, `simplePack` is $2b(d_v + 1)$ -approximate. Similarly, `betterSimplePack` has AAR $b(1 + \ln(d_v + 1) + \varepsilon)$, and we can get a $b(1 + \varepsilon)$ -approximation algorithm for (d_g, d_v) KS.

4 Round-and-Approx Framework

We enhance the R&A framework as a general outline for designing approximation algorithms for bin packing and its variants. We denote the algorithm for the R&A framework as `rnaPack`(I, β, ε), which takes as input a set I of (d_g, d_v) -dimensional items and parameters $\beta \geq 1$ and $\varepsilon \in (0, 1)$. The steps of the algorithm are as follows. (See Algorithm 1 in Appendix C for a more formal description).

1. **Solve the Configuration LP of I :** Let \hat{x} be a μ -asymptotic-approximate solution to the configuration LP. Note that each index of \hat{x} corresponds to a configuration. In all previous applications of R&A, $\mu = 1 + \varepsilon$, but in our work, μ can be a large constant.
2. **Randomized rounding of configuration LP:** For $T := \lceil (\ln \beta) \|\hat{x}\|_1 \rceil$ steps do the following: select a configuration C with probability $\hat{x}_C / \|\hat{x}\|_1$. Pack T bins according to each of these selected T configurations. Let S be the remaining items which are not packed, called the *residual instance*.
3. **Rounding of items:** We define a subroutine **round** that takes items I and parameter ε as input¹. It *discards* a set $D \subseteq I$ of items such that $\text{span}(D) \leq \varepsilon \text{span}(I)$ and then *modifies* each item in $I - D$ to get a set \tilde{I} of items. We say that the output of **round**(I, ε) is (\tilde{I}, D) , where items in \tilde{I} are called *rounded items*. Intuitively, after rounding, the items in \tilde{I} are of $O(1)$ types, which makes packing easier. Also, since $\text{span}(D)$ is small, $D \cap S$ can be packed into a small number of bins using **simplePack**.
We impose some restrictions on **round**, which we denote as conditions C1 and C2, which we describe in Section 4.2. Previous versions of R&A only allowed modifications where items' dimensions were rounded up. We don't have this restriction; we also allow rounding down some dimensions. We also allow **round** to output a poly(n)-sized list of guesses for (\tilde{I}, D) .
4. **Pack rounded items:** Let \tilde{S} be the rounded items corresponding to $S \setminus D$. Pack \tilde{S} into bins using any bin packing algorithm that satisfies "condition C3", which we describe in Section 4.3. Let us name this algorithm **complexPack**.
5. **Unrounding:** Given a bin packing of \tilde{S} , let **unround** be a subroutine that computes a bin packing of $S \setminus D$. **unround** is trivial in previous versions of R&A, because they only increase dimensions of items during rounding. In our applications, we may round down items, so **unround** can be non-trivial. **unround** can be any algorithm that satisfies "condition C4", which we describe in Section 4.3.

We can think of the R&A framework as a *meta-algorithm*, i.e., we give it the algorithms **round**, **complexPack** and **unround** as inputs and it outputs the algorithm **rnaPack**. The R&A framework requires that **round**, **complexPack** and **unround** satisfy four conditions C1, C2, C3, C4, which we describe in Sections 4.2 and 4.3. Prospective users of the R&A framework need to design these three subroutines and prove that they satisfy these four conditions.

Intuitively, **rnaPack** first packs some items into T bins by randomized rounding of \hat{x} . We can prove that $\Pr(i \in S) \leq 1/\beta$ (using standard techniques from randomized rounding, see Lemma 16 in Appendix C), so S contains a small fraction of the items in I . We will then try to prove that if the rest of the algorithm (**round** + **complexPack** + **unround**) packs I into m bins, then it will pack S into roughly m/β bins. This notion was referred to in [3] as *subset-obliviousness*. We will use subset-obliviousness to bound the AAR of **rnaPack**. Section 4.5 shows how to break **simplePack** into **round**, **complexPack**, and **unround** and use it with R&A.

¹ The input to **round** is I instead of S because S is random and we want to round items deterministically, i.e., the rounding of each item $i \in S$ should not depend on which other items from I lie in S . In fact, this is where the old R&A framework [35] introduced an error. See Appendix C.1 for details.

4.1 Fractional Structured Packing

Let (\tilde{I}, D) be an output of $\text{round}(I)$ and let \tilde{X} be an arbitrary subset of \tilde{I} . Our analysis of `rnaPack` is based around a concept called *fractional structured packing* of \tilde{X} . Note that the notion of fractional structured packing only appears in the analysis of `rnaPack`. It is not needed to describe any algorithm.

First, we discuss the notion of structured packing. Several types of packings are used in bin packing algorithms, such as *container-based* [27], *shelf-based* [14, 10], *guillotine-based* [24], *corridor-based* [20], *\mathcal{N} -box & \mathcal{V} -box* based [26], etc. A type of packing is called a *structured packing* if it satisfies *downward closure*, i.e., a *structured packing remains structured even after removing some items from the packed bins*. For example, Jansen and Prädél [27] showed that given any packing of a 2-D GBP instance into m bins, we can slice some of the items and repack them into $(1.5 + \varepsilon)m + O(1)$ bins such that the resulting packing is *container-based*. *Container-based* roughly means that in each bin, items are packed into rectangular regions called containers, and containers' heights and widths belong to a fixed set of $O(1)$ values. Hence, *container-based* is an example of a structured packing as a container-based packing remains container-based even after removing some items from the packed bins. Also note that the set of all possible packings is trivially a structured packing. Our R&A framework gives algorithm designers the freedom to use or define structured packing in any way they want, as long as they satisfy downward closure. Typically, the choice of the definition of structured packing will depend on the ease of proving Conditions C2 and C3 for that definition. This helped us go beyond Bansal and Khan's R&A framework [6], which only considered container-based packings.

Intuitively, a fractional structured packing is one where we slice each item of \tilde{X} into pieces and then find a structured packing of the pieces. Let $\text{fsopt}(\tilde{X})$ be the number of bins in the optimal fractional structured packing of \tilde{X} . To analyze the AAR of `rnaPack`, we will bound $\text{complexPack}(S)$ in terms of $\text{fsopt}(S)$, and then bound $\text{fsopt}(S)$ in terms of $\text{opt}(I)$.

To define fractional structured packing, we first define what it means to slice an item. From a geometric perspective, slicing an item perpendicular to the k^{th} dimension means cutting the item into 2 parts using a hyperplane perpendicular to the k^{th} axis. The vector dimensions get split proportionately across the slices. E.g., for $d_g = 2$, if $k = 1$ for item i , then we slice i using a vertical cut, and if $k = 2$, we slice i using a horizontal cut.

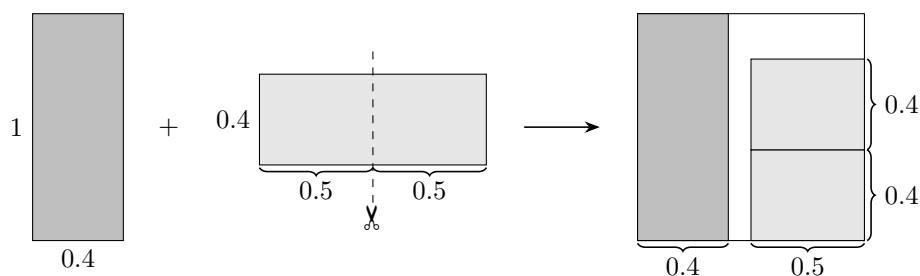
► **Definition 4** (Slicing an item). *Let i be a (d_g, d_v) -dimensional item. Slicing i perpendicular to geometric dimension k with proportionality α (where $0 < \alpha < 1$) is the operation of replacing i by two items i_1 and i_2 such that: (i) $\forall j \neq k, \ell_j(i) = \ell_j(i_1) = \ell_j(i_2)$, (ii) $\ell_k(i_1) = \alpha \ell_k(i)$ and $\ell_k(i_2) = (1 - \alpha) \ell_k(i)$, (iii) $\forall j \in [d_v], v_j(i_1) = \alpha v_j(i)$ and $v_j(i_2) = (1 - \alpha) v_j(i)$.*

► **Definition 5** (Fractional packing). *Let \tilde{I} be (d_g, d_v) -dimensional items, where for each item $i \in \tilde{I}$, we are given a set $X(i)$ of axes perpendicular to which we can repeatedly slice i ($X(i)$ can be empty, which would mean that the item cannot be sliced). If we slice items as per their given axes and then pack the slices into bins, then the resulting packing is called a fractional bin packing. (See Figure 1 for an example.)*

4.2 Properties of `round`

► **Definition 6.** *The density vector of a (d_g, d_v) item i is the vector $v_{\text{span}} := [v_0(i)/\text{span}(i), v_1(i)/\text{span}(i), \dots, v_{d_v}(i)/\text{span}(i)]$. Recall that $v_0(i) := \text{vol}(i)$.*

The subroutine `round`(I) returns a set of pairs of the form (\tilde{I}, D) . **Condition C1** is defined as the following constraints over each pair (\tilde{I}, D) :



■ **Figure 1** Example of a fractional packing of two items into a bin.

- **C1.1.** *Small discard:* $D \subseteq I$ and $\text{span}(D) \leq \varepsilon \text{span}(I)$.
- **C1.2.** *Bijection from $I - D$ to \tilde{I} :* Each item in \tilde{I} is obtained by modifying an item in $I - D$. Let π be the corresponding bijection from $I - D$ to \tilde{I} .
- **C1.3.** *Homogeneity properties:* round partitions items in \tilde{I} into a constant number of classes: $\tilde{K}_1, \tilde{K}_2, \dots, \tilde{K}_q$. These classes should satisfy the following properties, which we call *homogeneity* properties:
 - All items in a class have the same density vector.
 - For each class \tilde{K}_j , we decide the set X of axes perpendicular to which we can slice items in \tilde{K}_j . If items in a class \tilde{K}_j are not allowed to be sliced perpendicular to dimension k , then all items in that class have the same length along dimension k . (For example, if $d_g = 2$ and only vertical cuts are forbidden, then all items have the same width. However, they can have different heights.)
- **C1.4.** *Bounded expansion:* Let C be any configuration of I and \tilde{K} be any one of the constant number of classes of \tilde{I} . Let $\tilde{C} := \{\pi(i) : i \in C - D\}$. Then we need to prove that $\text{span}(\tilde{K} \cap \tilde{C}) \leq c_{\max}$ for some constant c_{\max} .

Intuitively, the homogeneity properties allow us to replace (a slice of) an item in a fractional packing by slices of other items of the same class. Thus, while trying to get a fractional packing, we can focus on the item classes, which are constant in number, instead of focusing on the n items. Intuitively, bounded expansion (C1.4) ensures that we do not round up items too much.

Condition C2 (also called *structural theorem*): For some constant $\rho > 0$ and for some $(\tilde{I}, D) \in \text{round}(I)$, $\text{fsopt}(\tilde{I}) \leq \rho \text{opt}(I) + O(1)$.

Intuitively, the structural theorem says that allowing slicing as per **round** and imposing a structure on the packing does not increase the minimum number of bins by too much. We will see that **rnaPack**'s AAR increases with ρ , so we want ρ to be small.

4.3 complexPack and unround

- **Condition C3:** For some constant $\alpha > 0$ and for any $(\tilde{I}, D) \in \text{round}(I)$ and any $\tilde{X} \subseteq \tilde{I}$, $\text{complexPack}(\tilde{X})$ packs \tilde{X} into at most $\alpha \text{fsopt}(\tilde{X}) + O(1)$ bins.
- **Condition C4:** For some constant $\gamma > 0$, if $\text{complexPack}(\tilde{S})$ outputs a packing of \tilde{S} into m bins, then **unround** converts that to a packing of $S - D$ into $\gamma m + O(1)$ bins.

Intuitively, Condition C3 says that we can find a packing of the rounded items that is close to the optimal fractional structured packing. Condition C4 says that unrounding does not increase the number of bins by too much. We will see that **rnaPack**'s AAR increases with α and γ , so we want α and γ to be small. If **round** only increases the dimensions of items, then unrounding is trivial and $\gamma = 1$.

4.4 AAR of R&A

Recall that `simplePack` is a $2b(d_v+1)$ -approximation algorithm for (d_g, d_v) BP (see Section 3). Our key ingredient in the analysis of R&A is the following lemma. We give a very brief outline of the proof here and defer the full proof to Appendix C.

► **Lemma 7.** *Let \tilde{S} be as computed by `rnaPack` (I, β, ε) . Then with high probability, we get $\text{fsopt}(\tilde{S}) \leq \text{fsopt}(\tilde{I})/\beta + 2b\mu\varepsilon \text{opt}(I) + O(1/\varepsilon^2)$.*

Proof sketch. Our proof of Lemma 7 is inspired by the analysis in [35]. We prove it by analyzing the *fractional structured configuration LP* of \tilde{I} .

► **Definition 8.** *Let $(\tilde{I}, D) \in \text{round}(I)$. Suppose `round` partitioned \tilde{I} into classes $\tilde{K}_1, \dots, \tilde{K}_q$. Let \mathcal{C}_f be the set of all structured configurations of items in \tilde{I} that allow items to be sliced as per `round`. For any $\tilde{S} \subseteq \tilde{I}$, the fractional structured configuration LP of \tilde{S} , denoted as $\text{fsLP}(\tilde{S})$, is*

$$\min_{x \in \mathbb{R}_{\geq 0}^{|\mathcal{C}_f|}} \sum_{C \in \mathcal{C}_f} x_C \quad \text{where} \quad \sum_{C \in \mathcal{C}_f} \text{span}(C \cap \tilde{K}_j) x_C \geq \text{span}(\tilde{S} \cap \tilde{K}_j) \quad \forall j \in [q]$$

The integer version of this program is called $\text{fsIP}(\tilde{S})$. The optimal objective values of $\text{fsLP}(\tilde{S})$ and $\text{fsIP}(\tilde{S})$ are denoted as $\text{fsLP}^*(\tilde{S})$ and $\text{fsIP}^*(\tilde{S})$, respectively.

Intuitively, fsIP is the same as the structured fractional bin packing problem because of the downward closure property and homogeneity, so $\text{fsIP}^*(\tilde{S}) \approx \text{fsopt}(\tilde{S})$ (see Lemma 17 in Appendix C for the proof). By homogeneity (C1.3), the number of constraints in this LP is a constant q . So, by Rank Lemma², we can show that $|\text{fsopt}(\tilde{S}) - \text{fsLP}^*(\tilde{S})| \in O(1)$ (see Lemma 18 in Appendix C for the proof). Now to prove Lemma 7, roughly, we need to show that $\text{fsLP}^*(\tilde{S}) \lesssim \text{fsLP}^*(\tilde{I})/\beta$.

The RHS in the j^{th} constraint of $\text{fsLP}(\tilde{S})$ is a random variable $\text{span}(\tilde{S} \cap \tilde{K}_j)$. The RHS in the j^{th} constraint of $\text{fsLP}(\tilde{I})$ is $\text{span}(\tilde{K}_j)$. Now using properties of randomized rounding, one can show $\forall i \in I, \Pr(i \in S) \leq 1/\beta$. (see Lemma 16 in Appendix C for the proof). Using this, we obtain $\mathbb{E}(\text{span}(\tilde{S} \cap \tilde{K}_j)) \leq \text{span}(\tilde{K}_j)/\beta$. In fact, we can harness the randomness of \tilde{S} , the bounded expansion property (C1.4), and McDiarmid's inequality [41] to show that $\text{span}(\tilde{S} \cap \tilde{K}_j) \lesssim \text{span}(\tilde{K}_j)/\beta$. Therefore, if x^* is an optimal solution to $\text{fsLP}(\tilde{I})$, then x^*/β is *roughly* a solution to $\text{fsLP}(\tilde{S})$, which implies $\text{fsLP}^*(\tilde{S}) \lesssim \text{fsLP}^*(\tilde{I})/\beta$ (see Lemma 21 in Appendix C for details). ◀

► **Theorem 9.** *With high probability, the number of bins used by `rnaPack` (I, β, ε) is at most $((\ln \beta)\mu + (\gamma\alpha\rho)/\beta + 2b(d_v + 1 + \gamma\alpha\mu)\varepsilon) \text{opt}(I) + O(1/\varepsilon^2)$.*

Proof sketch. (See Appendix C for full proof)

- We use at most $T \leq (\ln \beta)\mu \text{opt}(I) + O(1)$ bins to pack $I - S$.
- We use at most $b \lceil 2 \text{span}(D) \rceil \leq 2b(d_v+1)\varepsilon \text{opt}(I) + b$ bins to pack $S \cap D$ using `simplePack`.
- $S - D$ occupies at most $\gamma\alpha \text{fsopt}(\tilde{S}) + O(1) \leq \gamma\alpha (\rho/\beta + 2b\mu\varepsilon) \text{opt}(I) + O(1/\varepsilon^2)$ bins by Lemma 7 and Conditions C2, C3, and C4. ◀

Thus, the AAR of `rnaPack` (I) is roughly $\mu \ln \beta + \gamma\alpha\rho/\beta$. This is minimized for $\beta = \gamma\alpha\rho/\mu$ and the minimum value is $\mu(1 + \ln(\alpha\gamma\rho/\mu))$. As we require $\beta \geq 1$, we get this AAR only when $\gamma\alpha\rho \geq \mu$. If $\mu \geq \gamma\alpha\rho$, the optimal β is 1 and the AAR is roughly $\gamma\alpha\rho$.

² Rank Lemma: the number of non-zero variables in an extreme point of the set $\{x : Ax \geq b, x \geq 0\}$ is at most $\text{rank}(A)$. See Lemma 2.1.4 in [39].

4.5 Example: simplePack

We will show how to use `simplePack` with the R&A framework. Recall that `simplePack` is a $2b(d_v + 1)$ -approximation algorithm for (d_g, d_v) BP (see Section 3). Using the R&A framework on `simplePack` will improve its AAR from $2b(d_v + 1)$ to $b(1 + \ln(2(d_v + 1))) + O(\varepsilon)$. To do this, we need to show how to implement `round`, `complexPack`, and `unround`.

1. `solveConfigLP(I)`: Using the (d_g, d_v) KS algorithm of Section 3 and the LP algorithm of [47], we get a $b(1 + \varepsilon)$ -approximate solution to `configLP(I)`. Therefore, $\mu = b(1 + \varepsilon)$.
2. `round(I)`: returns just one pair: $(\tilde{I}, \{\})$, where $\tilde{I} := \{\pi(i) : i \in I\}$ and $\pi(i)$ is an item having height (i.e., d_g^{th} geometric dimension) equal to $\text{span}(i)$, all other geometric dimensions equal to 1, and all vector dimensions equal to $\text{span}(i)$. There is just one class in \tilde{I} , and we allow all items to be sliced perpendicular to the height, so the homogeneity properties are satisfied. Also, $c_{\max} = d_v + 1$ by Lemma 1 (since for any configuration C , we have $\text{span}(\pi(C)) = \text{span}(C) \leq (d_v + 1) \text{opt}(C) = d_v + 1$).
3. **Structural theorem**: We take structured packing to be the set of all possible packings. We can treat \tilde{I} as the 1-D instance $\{\text{span}(i) : i \in I\}$, so $\text{fsopt}(\tilde{I}) = \lceil \text{span}(I) \rceil \leq (d_v + 1) \text{opt}(I)$, where the inequality follows from Lemma 1. So, $\rho = d_v + 1$.
4. `complexPack(S)`: We can treat \tilde{S} as the 1-D instance $\{\text{span}(i) : i \in S\}$ and pack it using Next-Fit [30]. Hence, $|\text{complexPack}(\tilde{S})| \leq \lceil 2 \text{span}(S) \rceil \leq 2 \lceil \text{span}(S) \rceil = 2 \text{fsopt}(\tilde{S})$. So, $\alpha = 2$.
5. `unround(J)`: We are given a packing \tilde{J} of items \tilde{S} . For each bin in \tilde{J} , we can pack the corresponding unrounded items into b bins. Therefore, $\gamma = b$.

Hence, we get an AAR of $\mu(1 + \ln(\gamma\alpha\rho/\mu)) + O(\varepsilon) \approx b(1 + \ln(2(d_v + 1))) + O(\varepsilon)$.

For $d_g = 2$, we can slightly improve the AAR by using the $(2 + \varepsilon)$ -approximation algorithm of [37] for $(2, d_v)$ KS. This gives us an AAR of $2(1 + \ln(3(d_v + 1))) + O(\varepsilon)$. This is better than the AAR of `betterSimplePack` for $d_v \geq 3$.

The above example is presented only to illustrate an easy use of the R&A framework. It doesn't exploit the full power of the R&A framework. The algorithm `cbPack`, which we describe in Section 5, uses more sophisticated subroutines `round`, `complexPack` and `unround`, and uses a more intricate definition of fractional structured packing to get an even better AAR of $2(1 + \ln(\frac{d+4}{2})) + \varepsilon$ (improves to $2(1 + \ln(19/12)) + \varepsilon \approx 2.919 + \varepsilon$ for $d = 1$).

5 Improved Approximation Algorithms

In this section, we give an overview of the `cbPack` algorithm for $(2, d)$ BP, which is inspired from the 1.5 asymptotic approximation algorithm for 2-D GBP [44]. `cbPack` is based on the following two-step procedure, as is common in many packing algorithms.

In the first step (*structural step*, Appendices F.1–F.5 and F.7 in [38]), we show the existence of a good *structured* solution. Formally, we show that if items I can be packed into m bins, then we can round I to get a new instance \tilde{I} such that $\text{fsopt}(\tilde{I}) \leq \rho m + O(1)$ for some constant ρ , where $\text{fsopt}(\tilde{I})$ is the number of bins in the optimal *structured fractional* packing of I . Roughly, the notion of *structured* packing that we use here, which we call *compartmental packing*, imposes the following additional constraints over the *container-based* packing of [44]: (i) An item i is called *dense* iff $v_{\max}(i)/a(i)$ is above a certain threshold. If a bin contains dense items, then we reserve a sufficiently-large rectangular region exclusively for them. (ii) For a constant ε , for every $j \in [d]$, the sum of v_j of items in each bin is at most $1 - \varepsilon$. The proof of our structural result differs significantly from that of [44] because the presence of vector dimensions inhibit a straightforward application of their techniques.

In the second step (*algorithmic step*), we show how to find a near-optimal structured solution. `cbPack` first rounds I to \tilde{I} and then uses brute-force and LP to pack the items into containers, similar to [44] or [34]. Then we convert that packing to a non-fractional packing of I with only a tiny increase in the number of bins (see Appendix F.8 of [38] for the algorithmic step).

Then we show that `cbPack` fits into the R&A framework (i.e., components from `cbPack` can be used as `round`, `complexPack` and `unround`) and gives an AAR of roughly $2(1 + \ln(\rho/2))$. To (approximately) solve the configuration LP, we use the LP algorithm from [47] and the $(2 + \varepsilon)$ -approximation algorithm for $(2, d)$ KS from [37] (see Appendix F.9 of [38] for the details of fitting `cbPack` into the R&A framework).

5.1 Overview of Structural Result

We now give a brief overview of some key ideas used in our structural result. Due to space limitations, the details of the structural result and the algorithm can be found in Appendix F of [38]. Like [44], we start with a packing of input I into m bins, and transform it into a structured fractional packing of \tilde{I} into $\rho m + O(1)$ bins. To do this, just like many other papers on packing, we first classify the items into different classes based on their geometric and vector dimensions, and densities as follows. First, we identify two constants (which depend on ε) $\varepsilon_1, \varepsilon_2 \in (0, 1)$ such that $\varepsilon_1 > \varepsilon_2$ and the set of medium items defined as

$$I_{\text{med}} := \left\{ i \in I : w(i) \in (\varepsilon_2, \varepsilon_1] \vee h(i) \in (\varepsilon_2, \varepsilon_1] \vee \left(\bigvee_{j=1}^d v_j(i) \in (\varepsilon_2, \varepsilon_1] \right) \right\}$$

has a very small span, i.e., $\text{span}(I_{\text{med}}) \leq \varepsilon \text{span}(I)$. Hence, we can treat the items in I_{med} separately as they can be packed in a very small number of bins. Also, $\varepsilon_1, \varepsilon_2$ satisfy that $\varepsilon_2 \leq \varepsilon_1^2 \varepsilon / 2$. Informally, ε_2 is very small when compared to ε_1 . Hence, every item in $I \setminus I_{\text{med}}$ has the property that both width and height are big ($> \varepsilon_1$) or both width and height are small ($\leq \varepsilon_2$) or it is skewed. Based on $\varepsilon_1, \varepsilon_2$, we classify every item $i \in I \setminus I_{\text{med}}$ as follows. First, we classify by geometric dimensions as follows.

- Big item: $w(i) > \varepsilon_1$ and $h(i) > \varepsilon_1$.
- Wide item: $w(i) > \varepsilon_1$ and $h(i) \leq \varepsilon_2$.
- Tall item: $w(i) \leq \varepsilon_2$ and $h(i) > \varepsilon_1$.
- Small item: $w(i) \leq \varepsilon_2$ and $h(i) \leq \varepsilon_2$.

Then, we perform another classification depending on vector dimensions.

- **Definition 10** (Dense items). *Item i is dense iff either $a(i) = 0$ or $v_{\max}(i)/a(i) > 1/\varepsilon_1^2$.*
- **Definition 11** (Heavy and light items). *A dense item i is said to be heavy in vector dimension j iff $v_j(i) \geq \varepsilon_1$. Otherwise i is said to be light in dimension j . If i is heavy in some dimension, then i is said to be heavy, otherwise i is light.*

More formal details of this classification are provided in Appendix F.1 of [38].

Then, the structural result is obtained in three steps:

- (i) In the first step, we round up one geometric dimension of each item and pack the items into roughly $\rho m + O(1)$ bins. We call these bins *quarter-structured* (see Appendices F.2 and F.3 of [38]).
- (ii) In the second step, we round the remaining dimensions of items and partition them into classes such that they satisfy the *homogeneity properties* (see Section 4.2). We allow slicing and repack the items into almost the same number of bins. We call the resulting bin packing *semi-structured* (see Appendices F.4 and F.5 of [38]).

- (iii) In the third and final step, we transform the packing into a compartmental packing (see Appendices F.6 and F.7 of [38]). Compartmental packings have nice properties which help us find them efficiently. Roughly, a compartmental packing is a semi-structured packing that is also container-based. This step is very similar to [44].

In steps (i), (ii) and (iii), [44] uses the Next-Fit-Decreasing-Height (NFDH) algorithm [14] to pack items of $O(\epsilon m)$ area into $O(\epsilon m)$ bins. This does not work when vector dimensions are present as an item of low area can have large weights. In step (ii), [44] uses *linear grouping*, i.e., each item is moved in place of a geometrically larger item so that it can be rounded up. Vector dimensions make such cross-bin movement difficult, since that can violate bins' weight capacities. [44] uses cross-bin movement in step (i) too.

We first observe that most difficulties associated with vector dimensions disappear if items' *density* is upper-bounded by a constant. Density of item i is defined as $v_{\max}(i)/a(i)$. Specifically, if items of bounded density (we call them *non-dense* items) have small area, then `simplePack` can pack them into a small number of bins. Linear grouping can also be made to work for such items with some more effort. Hence, we segregate items as *dense* and *non-dense*. We reserve a thin rectangular region in bins for dense items, and the rest is reserved for non-dense items.

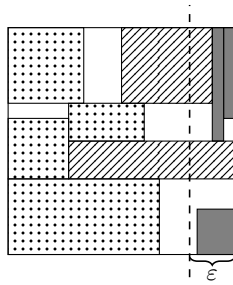
Furthermore, dense items in a bin must have low total area, due to their high density. If we reserve enough space for them in the bin, we can always pack them in their reserved region using NFDH (see Lemma 15 in Appendix B). Such a guarantee means that we can essentially ignore their geometric dimensions and simply treat them as vectors.

In step (ii), we want to round up vector dimensions with only a marginal increase in the number of bins. To do this, we require each quarter-structured bin to be ϵ -slacked. ϵ -slackness roughly means that for a set J of items in a bin, $\forall j \in [d], v_j(J) \leq 1 - \epsilon$ (see Appendix F.3 of [38] for a formal description). ϵ -slackness also helps us use existing algorithms for resource-augmented vector bin packing as subroutines. Also, during the rounding step, we round down the weight of some dense items, and ϵ -slackness allows us to *unround* with no increase in the number of bins.

The observations above guide our definition of *quarter-structured*. Roughly, a packing is quarter-structured if items having large width have their width and x -coordinate rounded to a multiple of $\epsilon_1^2/4$ and each bin is ϵ -slacked. We reserve a thin rectangular region of width $\epsilon_1/2$ for packing dense items (only if the bin contains dense items).

In step (i), [44] uses a standard cutting-strip argument: They create a strip of width ϵ_1 next to an edge of the bin (see Figure 2). Items completely inside the strip (called dark items), have small area and are packed separately using NFDH. Items intersecting the strip's boundary (called shaded items) are removed. This creates an empty space of width ϵ_1 in the bin. Using this empty space, items outside the strip (called dotted items) can then have their width and x -coordinate rounded to a multiple of $\epsilon_1^2/2$. Their key idea is how to pair up most bins so that shaded items from two bins can be rounded and packed together into a new bin. This is roughly why they get an AAR of $1.5 + \epsilon$.

We use the cutting-strip argument too, but with some differences. We cannot freely mix shaded items from different bins if they have large weight, and we cannot simply pack dark items into a small number of bins. We also need bins to be slacked. So, we get a larger AAR of $d + 4 + \epsilon$. For $d = 1$, however, we allow mixing items using more sophisticated techniques, which improves the AAR to $19/6 + \epsilon$. Also, we round dotted items to a multiple of $\epsilon_1^2/4$ instead of $\epsilon_1^2/2$, which leaves an empty strip of width $\epsilon_1/2$ in the bin even after rounding, and we reserve this space for dense items. This gives us a quarter-structured packing.



■ **Figure 2** Example of classifying items as dark, shaded and dotted based on an ε -strip.

Based on the broad ideas above, we make more changes to the quarter-structured packing to get a compartmental packing. Finally, in the algorithmic step, we use a combination of brute-force and LP to pack items into compartments and efficiently find a *good* compartmental packing. An overview is given in the section below.

5.2 Overview of the Algorithmic Step

Note that in any bin, the number of big items and heavy items can be at most a constant in number. The nice structure obtained in the structural step has the property that in each bin there are constant number of compartments in which all the light items (which are not big) are packed. With these arguments, in any bin of this nice structure, the number of big items, heavy items, and compartments is constant in number. Moreover, the sizes of the compartments also belong to a polynomial sized set. There are other things to be considered (e.g., the slack type of the bin), but one of the main implications of the structural result is that the number of possible *configurations* of each bin is at most a constant. Also, note that the number of bins itself is bounded by the number of items n . Thus, in polynomial time (by brute force), we can guess the number of bins in the nice structure, and the configuration of each bin (for full details, see Appendix F.8.1 of [38]).

Once the configurations have been guessed, we are left with the task of packing the wide, tall, small, and light items in the compartments. First, as an intermediate step, we obtain a fractional packing of these items into the compartments by solving a linear program (if the linear program is infeasible, we realize that our guess of the bin configurations is incorrect). See Appendix F.8.2 of [38] for the details of the linear program. The main purpose of solving the linear program (if it turns out to be feasible) is to further divide compartments into what are called *containers* (see Appendix F.8.3 of [38]). Finally, the wide, tall, small, and light items are non-fractionally packed into the containers greedily. There can be some items which can't be packed by our greedy strategy, but we can prove that their span is so small that we can pack them using very few extra bins. See Appendices F.8.5, F.8.6, and F.8.6 of [38] for the details of packing the wide, tall, small, and light items.

In Theorem 66 of Appendix F.8.7 of [38], we compute the approximation factor of this algorithm:

► **Theorem 12.** *For general d , when rotations are forbidden, the above algorithm packs the input set I into at most*

$$(1 + 23\varepsilon)(d + 4) \text{opt}(I) + O_\varepsilon(1)$$

number of bins. Here $O_\varepsilon(1)$ hides a constant that depends on ε .

In the special case of $d = 1$, we obtain a better approximation ratio.

► **Theorem 13.** *If $d = 1$ and rotations are forbidden, we can pack the input set I into at most*

$$(1 + 23\varepsilon) \left(3 + \frac{1}{6} + \frac{\varepsilon}{1 - \varepsilon} \right) \text{opt}(I) + O_\varepsilon(1)$$

number of bins. Here $O_\varepsilon(1)$ hides a constant that depends on ε .

Using similar techniques, we can design an algorithm for the case with rotations too. Using the Round&Approx framework on top of this algorithm, and scaling ε appropriately, we get the following approximation ratios.

- For general d , with no rotations, we get an approximation ratio of $2 \left(1 + \ln \left(\frac{d+4}{2} \right) \right) + \varepsilon$.
- When $d = 1$ and rotations are forbidden, we get an approximation ratio of $\approx 2.919 + \varepsilon$.
- For general d , with rotations allowed, we get an approximation ratio of $2 \left(1 + \ln \left(\frac{d+3}{2} \right) \right) + \varepsilon$.
- When $d = 1$ and rotations are allowed, we get an approximation ratio of $\approx 2.811 + \varepsilon$.

References

- 1 M. T. Alonso, R. Alvarez-Valdes, Manuel Iori, F. Parreño, and J. M. Tamarit. Mathematical models for multicontainer loading problems. *Omega*, 66:106–117, 2017. doi:10.1016/j.omega.2016.02.002.
- 2 Samir V. Amiouny, John J. Bartholdi III, John H. Vande Vate, and Jixian Zhang. Balanced loading. *Operations Research*, 40(2):238–246, 1992. doi:10.1287/opre.40.2.238.
- 3 Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM Journal on Computing*, 39(4):1256–1278, 2009. doi:10.1137/080736831.
- 4 Nikhil Bansal, Jose R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31:31–49, 2006. doi:10.1287/moor.1050.0168.
- 5 Nikhil Bansal, Marek Eliáš, and Arindam Khan. Improved approximation for vector bin packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1561–1579. SIAM, 2016. doi:10.1137/1.9781611974331.ch106.
- 6 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.
- 7 Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 4955–4966, 2019.
- 8 Andreas Bortfeldt and Gerhard Wäscher. Constraints in container loading—a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013. doi:10.1016/j.ejor.2012.12.006.
- 9 Alberto Caprara. Packing 2-dimensional bins in harmony. In *FOCS*, pages 490–499, 2002. doi:10.1109/SFCS.2002.1181973.
- 10 Alberto Caprara. Packing d -dimensional bins in d stages. *Mathematics of Operations Research*, 33:203–215, 2008. doi:10.1287/moor.1070.0289.
- 11 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004. doi:10.1137/S0097539799356265.
- 12 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.
- 13 Edward G. Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Survey and classification. In *Handbook of combinatorial optimization*, pages 455–531. Springer New York, 2013. doi:10.1007/978-1-4419-7997-1_35.
- 14 Edward G. Coffman, Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980. doi:10.1137/0209062.

- 15 Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3D orthogonal knapsack. *Journal of Computer Science and Technology*, 23(5):749, 2008. doi:10.1007/s11390-008-9170-7.
- 16 Khaled M. Elbassioni, Naveen Garg, Divya Gupta, Amit Kumar, Vishal Narula, and Arindam Pal. Approximation algorithms for the unsplitable flow problem on paths and trees. In *FSTTCS*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 267–275, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.267.
- 17 Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981. doi:10.1007/BF02579456.
- 18 A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m -dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *EJOR*, 15:100–109, 1984.
- 19 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, Klaus Jansen, Arindam Khan, and Malin Rau. A tight $(3/2 + \epsilon)$ approximation for skewed strip packing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.44.
- 20 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 260–271. IEEE, 2017. Full version available at http://www.dii.uchile.cl/~awiese/2DK_full_version.pdf. doi:10.1109/FOCS.2017.32.
- 21 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan. Improved pseudo-polynomial-time approximation for strip packing. In *FSTTCS*, pages 9:1–9:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.9.
- 22 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A 3-approximation algorithm for maximum independent set of rectangles. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 894–905. SIAM, 2022. doi:10.1137/1.9781611977073.38.
- 23 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy, and Andreas Wiese. A $(2 + \epsilon)$ -approximation algorithm for maximum independent set of rectangles, 2021. arXiv:2106.00623.
- 24 Paul C Gilmore and Ralph E Gomory. Multistage cutting stock problems of two and more dimensions. *Operations research*, 13(1):94–120, 1965. doi:10.1287/opre.13.1.94.
- 25 Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *SODA*, pages 2616–2625, 2017. doi:10.1137/1.9781611974782.172.
- 26 Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A PTAS for packing hypercubes into a knapsack. In *49th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 229 of *LIPIcs*, pages 78:1–78:20, 2022.
- 27 Klaus Jansen and Lars Prädél. New approximability results for two-dimensional bin packing. *Algorithmica*, 74(1):208–269, 2016. doi:10.1007/s00453-014-9943-z.
- 28 Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *SODA*, pages 204–213, 2004.
- 29 D. S. Johnson. Approximation algorithms for combinatorial problems. In *STOC*, pages 38–49, 1973.
- 30 David S Johnson. *Near-Optimal Bin Packing Algorithms*. PhD thesis, Massachusetts Institute of Technology, USA, 1973.
- 31 Matthew Joseph, Michael J. Kearns, Jamie H. Morgenstern, and Aaron Roth. Fairness in learning: Classic and contextual bandits. In *NeurIPS*, pages 325–333, 2016.
- 32 Debajyoti Kar, Arindam Khan, and Andreas Wiese. Approximation algorithms for round-ufp and round-sap, 2022. doi:10.48550/ARXIV.2202.03492.
- 33 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

- 34 Claire Kenyon and Eric Rémila. Approximate strip packing. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 31–36, 1996. doi:10.1109/SFCS.1996.548461.
- 35 Arindam Khan. *Approximation algorithms for multidimensional bin packing*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2016.
- 36 Arindam Khan and Madhusudhan Reddy Pittu. On guillotine separability of squares and rectangles. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.47.
- 37 Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas. Approximation algorithms for generalized multidimensional knapsack, 2021. arXiv:2102.05854.
- 38 Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas. Geometry meets vectors: Approximation algorithms for multidimensional packing, 2021. arXiv:2106.13951v1.
- 39 Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.
- 40 Eugene L Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979. doi:10.1287/moor.4.4.339.
- 41 Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.
- 42 Célia Paquay, Michael Schyns, and Sabine Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, 2016. doi:10.1111/itor.12111.
- 43 Deval Patel, Arindam Khan, and Anand Louis. Group fairness for knapsack problems. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1001–1009, 2021.
- 44 Lars Dennis Prädell. *Approximation Algorithms for Geometric Packing Problems*. PhD thesis, Kiel University, 2012. URL: https://macau.uni-kiel.de/servlets/MCRFileNodeServlet/dissertation_derivate_00004634/dissertation-praedel.pdf?AC=N.
- 45 Arka Ray. There is no APTAS for 2-dimensional vector bin packing: Revisited, 2021. arXiv:2104.13362.
- 46 Sai Sandeep. Almost optimal inapproximability of multidimensional packing problems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 245–256, 2022. doi:10.1109/FOCS52979.2021.00033.
- 47 Eklavya Sharma. An approximation algorithm for covering linear programs and its application to bin-packing, 2020. arXiv:2011.11268.
- 48 Knut Olav Brathaug Sørset. A heuristic approach to the three-dimensional bin packing problem with weight constraints. Master’s thesis, Høgskolen i Molde-Vitenskapelig høgskole i logistikk, 2019.
- 49 A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997. doi:10.1137/S0097539793255801.
- 50 Gregory S. Taylor, Yupo Chan, and Ghulam Rasool. A three-dimensional bin-packing model: exact multicriteria solution and computational complexity. *Annals of Operations Research*, 251(1-2):397–427, 2017. doi:10.1007/s10479-015-2048-5.
- 51 Alan Tsang, Bryan Wilder, Eric Rice, Milind Tambe, and Yair Zick. Group-fairness in influence maximization. In *IJCAI*, pages 5997–6005, 2019.
- 52 Gerhard J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Inf. Process. Lett.*, 64(6):293–297, 1997. doi:10.1016/S0020-0190(97)00179-8.
- 53 Guang Yang. *gbp: a bin packing problem solver*, 2017. R package version 0.1.0.4. URL: <https://CRAN.R-project.org/package=gbp>.

A Formal Definition of (d_g, d_v) Packing

► **Definition 14.** A valid packing of (d_g, d_v) -dimensional items into a (d_g, d_v) -dimensional bin is an arrangement of the items in the bin such that all of the following hold:

1. All items are packed in an axis parallel manner, i.e., each item has its faces parallel to the faces of the bin. Formally, to pack item i into a bin, we need to decide its position $(x_1(i), x_2(i), \dots, x_{d_g}(i))$ in the bin, and the item will be packed in the cuboidal region $\prod_{j=1}^{d_g} [x_j(i), x_j(i) + \ell_j(i)]$.
2. The items are non-overlapping, i.e., the interiors of any two items do not intersect. Formally, for any two items i_1 and i_2 in the same bin, the sets $\prod_{j=1}^{d_g} (x_j(i_1), x_j(i_1) + \ell_j(i_1))$ and $\prod_{j=1}^{d_g} (x_j(i_2), x_j(i_2) + \ell_j(i_2))$ do not intersect.
3. All items are packed completely inside the bin, i.e., for each item i and each $j \in [d_g]$, $x_j(i) \geq 0$ and $x_j(i) + \ell_j(i) \leq 1$.
4. In each bin, the total weight in each of the d_v dimensions is at most one, i.e., for each set S of items in a bin and each $j \in [d_v]$, we have $\sum_{i \in S} v_j(i) \leq 1$.

B Next-Fit Decreasing Height (NFDH)

► **Lemma 15** (NFDH for strip packing [14]). A set I of rectangles can be packed into a strip of height at most $2a(I) + \max_{i \in I} h(i)$ using the Next-Fit Decreasing Height (NFDH) algorithm.

C Details of the R&A Framework

► **Algorithm 1** `rnaPack`(I, β, ε): Computes a bin packing of (d_g, d_v) items I , and $\beta \geq 1$.

```

1:  $\hat{x} = \text{solveConfigLP}(I)$ 
2: repeat  $T := \lceil (\ln \beta) \|\hat{x}\|_1 \rceil$  times
3:   Select a configuration  $C$  with probability  $\hat{x}_C / \|\hat{x}\|_1$ .
4:   Pack a bin according to  $C$ .
5: end repeat
6: Let  $S$  be the unpacked items from  $I$ . //  $S$  is called the set of residual items.
7: Initialize  $J_{\text{best}}$  to null.
8: for  $(\tilde{I}, D) \in \text{round}(I)$  do // round(I) outputs a set of pairs.
9:    $J_D = \text{simplePack}(S \cap D)$ 
10:  Let  $\pi$  be a bijection from  $I - D$  to  $\tilde{I}$ . Let  $\tilde{S} := \{\pi(i) : i \in S - D\}$ .
11:   $\tilde{J} = \text{complexPack}(\tilde{S})$ 
12:   $J = \text{unround}(\tilde{J})$ 
13:  if  $J_{\text{best}}$  is null or  $|J_D \cup J| < |J_{\text{best}}|$  then
14:     $J_{\text{best}} = J_D \cup J$ 
15:  end if
16: end for
17: Pack  $S$  according to  $J_{\text{best}}$ .

```

► **Lemma 16.** $\forall i \in I, \Pr(i \in S) \leq \exp\left(-\frac{T}{\|\hat{x}\|_1}\right) \leq \frac{1}{\beta}$.

Proof. Let C_1, C_2, \dots, C_T be the configurations chosen during randomized rounding (line 3 in Algorithm 1). Let \mathcal{C}_i be the configurations that contain the element i .

$$\begin{aligned} \Pr(i \in S) &= \Pr\left(\bigwedge_{t=1}^T (C_t \notin \mathcal{C}_i)\right) = \prod_{t=1}^T \Pr(C_t \notin \mathcal{C}_i) && \text{(all } C_t \text{ are independent)} \\ &= \prod_{t=1}^T \left(1 - \sum_{C \in \mathcal{C}_i} \Pr(C_t = C)\right) = \left(1 - \sum_{C \in \mathcal{C}_i} \frac{\hat{x}_C}{\|\hat{x}\|_1}\right)^T \\ &\leq \left(1 - \frac{1}{\|\hat{x}\|_1}\right)^T && \text{(constraint in configuration LP for item } i\text{)} \\ &\leq \exp\left(-\frac{T}{\|\hat{x}\|_1}\right) \leq \frac{1}{\beta} \end{aligned} \quad \blacktriangleleft$$

► **Lemma 17.** $\text{fsopt}(\tilde{S}) \leq \text{fsIP}^*(\tilde{S}) \leq \text{fsopt}(\tilde{S}) + q$.

Proof. Due to the downward closure property, changing inequality constraints to equality constraints doesn't affect the optimum values of the above LP and IP. Therefore, $\text{fsIP}(\tilde{S})$ is equivalent to the fractional structured bin packing problem.

The above definition of $\text{fsLP}(\tilde{I})$ is problematic: the number of variables may be infinite if some classes allow slicing. We circumvent this problem by *discretizing* the configurations: Let δ be the smallest dimension of any item, i.e. $\delta := \min\left(\min_{j=1}^{d_g} \ell_j(i), \min_{j=1}^{d_v} v_j(i)\right)$.

In any optimal integral solution to $\text{fsLP}(\tilde{I})$ that uses m bins, we can slice out some items from each class in each bin so that the span of each class in each bin is a multiple of δ^{d_g}/n . In each class, the total size of sliced out items across all bins is at most δ^{d_g} . Therefore, for each class, slices of that class can fit into a single item of that class. If each such single item is packed in a separate bin, the total number of bins used is at most $m + q$.

Therefore, we only need to consider configurations where either the span of each class is a multiple of δ^{d_g}/n or there is a single item in the configuration. This gives us a finite number of configurations and completes the proof. ◀

► **Lemma 18.** $\text{fsLP}^*(\tilde{S}) \leq \text{fsIP}^*(\tilde{S}) \leq \text{fsLP}^*(\tilde{S}) + q$.

Proof. By Rank Lemma (Lemma 2.1.4 in [39]), the number of positive-valued variables in an extreme-point solution to a linear program is at most the number of constraints (other than the variable non-negativity constraints).

Thus, an optimal extreme-point solution to $\text{fsLP}(\tilde{S})$ has at most q positive-valued variables. Rounding up those variables to the nearest integer will give us an integral solution and increase the objective value by at most q . Hence, $\text{fsIP}^*(\tilde{S}) \leq \text{fsLP}^*(\tilde{S}) + q$. ◀

Let $\text{configLP}(I)$ denote the configuration LP of items I and let $\text{configLP}^*(I)$ denote the optimal objective value of $\text{configLP}(I)$. Recall that **simplePack** is a $2b(d_v + 1)$ -approximation algorithm for (d_g, d_v) BP (see Section 3), where $b := 3$ when $d_g = 2$, $b := 9$ when $d_g = 3$, $b := 4^{d_g} + 2^{d_g}$ when $d_g > 3$, and $b := 1$ when $d_g = 1$.

► **Lemma 19.** For a set I of (d_g, d_v) -dimensional items, $\text{configLP}^*(I) \in \Theta(\text{span}(I)) + O(1)$.

Proof. Let A be the configuration matrix of I . Let x^* be the optimal solution to $\text{configLP}(I)$. In $\text{configLP}(I)$, the constraint for item i gives us $\sum_{C \in \mathcal{C}} A[i, C]x_C^* \geq 1$. Multiplying each constraint by $\text{span}(i)$ and adding these constraints together, we get

$$\begin{aligned} \text{span}(I) &\leq \sum_{C \in \mathcal{C}} \sum_{i \in I} \text{span}(i) A[i, C] x_C^* = \sum_{C \in \mathcal{C}} \text{span}(C) x_C^* \\ &\leq (d_v + 1) \sum_{C \in \mathcal{C}} x_C^* = (d_v + 1) \text{configLP}^*(I). \end{aligned}$$

Therefore, $\text{configLP}^*(I) \geq \text{span}(I)/(d_v + 1)$. We also have

$$\text{configLP}^*(I) \leq \text{opt}(I) \leq |\text{simplePack}(I)| \leq 2b \text{span}(I) + b.$$

Therefore, $\text{configLP}^*(I) \in \Theta(\text{span}(I)) + O(1)$. \blacktriangleleft

► **Lemma 20** (Independent Bounded Difference Inequality [41]). *Let $X := [X_1, X_2, \dots, X_n]$ be random variables with $X_j \in A_j$. Let $\phi : \prod_{i=1}^n A_j \mapsto \mathbb{R}$ be a function such that $|\phi(x) - \phi(y)| \leq c_j$ whenever vectors x and y differ only in the j^{th} coordinate. Then for any $t \geq 0$,*

$$\Pr[\phi(X) - \mathbb{E}(\phi(X)) \geq t] \leq \exp\left(-\frac{2t^2}{\sum_{j=1}^n c_j^2}\right).$$

► **Lemma 21.** *Let \tilde{S} be as computed by $\text{rnaPack}(I, \beta, \varepsilon)$. Let $\varepsilon \in (0, 1)$ be a constant. When $\text{span}(I)$ is large compared to $1/\varepsilon^2$, we get that with high probability*

$$\text{fsLP}^*(\tilde{S}) \leq \frac{\text{fsLP}^*(\tilde{I})}{\beta} + 2b\mu\varepsilon \text{opt}(I) + O(1).$$

Proof. Let $y \in \mathcal{C}^T$ be the configurations chosen during randomized rounding (on line 3 in rnaPack). When viewed as a vector of length T , all coordinates of y are independent. Define $\text{uncovered}(y) := I - \bigcup_{t=1}^T y_t$. Let S be the unpacked items from I (see line 6 in rnaPack). Then $S = \text{uncovered}(y)$.

Let $\tilde{K}_1, \dots, \tilde{K}_q$ be the classes of \tilde{I} . Let π be the bijection from $I - D$ to \tilde{I} . For a set $X \subseteq I$, let $\tilde{I}[X] := \{\pi(i) : i \in X - D\}$. For $j \in [q]$, define $\phi_j \in \mathcal{C}^T \mapsto \mathbb{R}_{\geq 0}$ as

$$\phi_j(y) := \text{span}\left(\tilde{K}_j \cap \tilde{I}[\text{uncovered}(y)]\right).$$

For any $X \subseteq I$, define $g_j(X) := \text{span}(\tilde{K}_j \cap \tilde{I}[X])$. Then $\phi_j(y) = g_j(\text{uncovered}(y))$ and g_j is a non-negative additive function.

Let $y^{(1)}, y^{(2)} \in \mathcal{C}^T$ such that $y^{(1)}$ and $y^{(2)}$ differ only in coordinate t . Let $C_1 := y_t^{(1)}$, $C_2 := y_t^{(2)}$, $S_1 := \text{uncovered}(y^{(1)})$, $S_2 := \text{uncovered}(y^{(2)})$, and $R := \bigcup_{t' \neq t} y_{t'}^{(1)} = \bigcup_{t' \neq t} y_{t'}^{(2)}$. Then $I - S_1 = R \cup C_1$ and $I - S_2 = R \cup C_2$. So, $S_1 - S_2 = (C_2 - C_1) - R \subseteq C_2$ and $S_2 - S_1 = (C_1 - C_2) - R \subseteq C_1$. Hence,

$$\begin{aligned} \left| \phi_j(y^{(1)}) - \phi_j(y^{(2)}) \right| &= |g_j(S_1) - g_j(S_2)| = |g_j(S_1 - S_2) - g_j(S_2 - S_1)| \quad (\text{additivity of } g_j) \\ &\leq \max(g_j(S_1 - S_2), g_j(S_2 - S_1)) \leq \max(g_j(C_2), g_j(C_1)) \\ &\leq \max_{C \in \mathcal{C}} \text{span}(\tilde{K}_j \cap \tilde{I}[C]) \leq c_{\max}. \quad (\text{by bounded expansion (C1.4)}) \end{aligned}$$

By Lemma 16, $\Pr(i \in S) \leq 1/\beta$. By linearity of \mathbb{E} and additivity of g_j , we get

$$\mathbb{E}(\phi_j(y)) = \mathbb{E}(g_j(S)) = \sum_{i \in \tilde{I}} g_j(\{i\}) \Pr(i \in S) \leq \sum_{i \in \tilde{I}} g_j(\{i\})(1/\beta) = \frac{g_j(\tilde{I})}{\beta} = \frac{\text{span}(\tilde{K}_j)}{\beta}.$$

$\forall j \in [q]$, define Q_j as the smallest prefix of $\tilde{S} \cap \tilde{K}_j$ such that either $Q_j = \tilde{S} \cap \tilde{K}_j$ or $\text{span}(Q_j) \geq \varepsilon \|\hat{x}\|_1 / q$. Define $Q := \bigcup_{j=1}^q Q_j$. Hence, $\text{span}(Q) \leq \varepsilon \|\hat{x}\|_1 + q \leq \varepsilon \mu \text{opt}(I) + O(1)$.

$$\begin{aligned} \text{fsLP}^*(\tilde{S}) &\leq \text{fsLP}^*(\tilde{S} - Q) + \text{fsLP}^*(Q) \\ &\leq \text{fsLP}^*(\tilde{S} - Q) + b(2\text{span}(Q) + 1) && \text{(by Section 3)} \\ &\leq \text{fsLP}^*(\tilde{S} - Q) + 2b\mu\varepsilon \text{opt}(I) + O(1). \end{aligned}$$

Now we will try to prove that with high probability, $\text{fsLP}^*(\tilde{S} - Q) \leq \text{fsLP}^*(\tilde{I})/\beta$.

If $Q_j = \tilde{S} \cap \tilde{K}_j$, then $\text{span}(\tilde{K}_j \cap (\tilde{S} - Q)) = 0$. Otherwise,

$$\begin{aligned} \Pr \left[\text{span}(\tilde{K}_j \cap (\tilde{S} - Q)) \geq \frac{\text{span}(\tilde{K}_j)}{\beta} \right] &= \Pr \left[\text{span}(\tilde{K}_j \cap \tilde{S}) - \frac{\text{span}(\tilde{K}_j)}{\beta} \geq \text{span}(Q_j) \right] \\ &\leq \Pr \left[\phi_j(y) - \mathbb{E}(\phi_j(y)) \geq \frac{\varepsilon}{q} \|\hat{x}\|_1 \right] \leq \exp \left(-\frac{2}{Tc_{\max}^2} \left(\frac{\varepsilon}{q} \|\hat{x}\|_1 \right)^2 \right) && \text{(Lemma 20)} \\ &\leq \exp \left(-\frac{2\varepsilon^2}{\ln(\beta)c_{\max}^2 q^2} \|\hat{x}\|_1 \right). \end{aligned}$$

Therefore, by union bound, we get

$$\Pr \left[\bigvee_{j=1}^q \left(\text{span}(\tilde{K}_j \cap (\tilde{S} - Q)) \geq \frac{\text{span}(\tilde{K}_j)}{\beta} \right) \right] \leq q \exp \left(-\frac{2\varepsilon^2}{\ln(\beta)c_{\max}^2 q^2} \|\hat{x}\|_1 \right).$$

Since $\text{configLP}^*(I) \leq \|\hat{x}\|_1 \leq \mu \text{configLP}^*(I) + O(1)$, and $\text{configLP}^*(I) \in \Theta(\text{span}(I)) + O(1)$ (by Lemma 19), we get $\|\hat{x}\|_1 \in \Theta(\text{span}(I)) + O(1)$. When $\text{span}(I)$ is very large compared to $1/\varepsilon^2$, we get that with high probability, $\forall j \in [q]$,

$$\text{span}(\tilde{K}_j \cap (\tilde{S} - Q)) \leq \frac{\text{span}(\tilde{K}_j)}{\beta}.$$

Let x^* be the optimal solution to $\text{fsLP}(\tilde{I})$. Then with high probability, x^*/β is a feasible solution to $\text{fsLP}(\tilde{S} - Q)$. Therefore,

$$\text{fsLP}^*(\tilde{S}) \leq \text{fsLP}^*(\tilde{S} - Q) + 2b\mu\varepsilon \text{opt}(I) + O(1) \leq \text{fsLP}^*(\tilde{I})/\beta + 2b\mu\varepsilon \text{opt}(I) + O(1). \quad \blacktriangleleft$$

► **Lemma 7.** *Let \tilde{S} be as computed by $\text{rnaPack}(I, \beta, \varepsilon)$. Then with high probability, we get $\text{fsopt}(\tilde{S}) \leq \text{fsopt}(\tilde{I})/\beta + 2b\mu\varepsilon \text{opt}(I) + O(1/\varepsilon^2)$.*

Proof. When $\text{span}(I)$ is very large compared to $1/\varepsilon^2$, we get

$$\begin{aligned} \text{fsopt}(\tilde{S}) &\leq \text{fsIP}^*(\tilde{S}) + O(1) && \text{(by Lemma 17)} \\ &\leq \text{fsLP}^*(\tilde{S}) + O(1) && \text{(by Lemma 18)} \\ &\leq \text{fsLP}^*(\tilde{I})/\beta + 2b\mu\varepsilon \text{opt}(I) + O(1) && \text{(by Lemma 21)} \\ &\leq \text{fsopt}(\tilde{I})/\beta + 2b\mu\varepsilon \text{opt}(I) + O(1). && \text{(by Lemma 17)} \end{aligned}$$

Otherwise, if $\text{span}(I) \in O(1/\varepsilon^2)$, we get

$$\begin{aligned} \text{fsopt}(\tilde{S}) &\leq \rho \text{opt}(I) + O(1) && \text{(by structural theorem)} \\ &\leq \rho |\text{simplePack}(I)| + O(1) \\ &\leq \Theta(\text{span}(I)) + O(1) \leq O(1/\varepsilon^2). && \text{(by Section 3)} \end{aligned}$$

◀

► **Theorem 9.** *With high probability, the number of bins used by `rnaPack`(I, β, ε) is at most $((\ln \beta)\mu + (\gamma\alpha\rho)/\beta + 2b(d_v + 1 + \gamma\alpha\mu)\varepsilon) \text{opt}(I) + O(1/\varepsilon^2)$.*

Proof. Let J_{LP} be the set of bins packed in the *randomized rounding of configuration LP* step (see line 4 in Algorithm 1 in Appendix C), J_D be the set of bins used to pack the discarded items $D \cap S$, J be the set of bins used to pack the rest of the items $S \setminus D$, and \tilde{J} be the set of bins used by `complexPack` to pack items in \tilde{S} .

Then $|J_{\text{LP}}| \leq T = \lceil (\ln \beta) \|\hat{x}\|_1 \rceil \leq (\ln \beta)\mu \text{opt}(I) + O(1)$.

Now, we have $|J_D| \leq b \lceil 2 \text{span}(D) \rceil \leq 2b\varepsilon \text{span}(I) + b \leq 2b(d_v + 1)\varepsilon \text{opt}(I) + b$. The first inequality follows from the property of `simplePack`, the second follows from C1.1 (Small Discard) and the last follows from Lemma 1. Finally,

$$\begin{aligned} |J| &\leq \gamma|\tilde{J}| + O(1) && \text{(property of unround (C4))} \\ &\leq \gamma\alpha \text{fsopt}(\tilde{S}) + O(1) && \text{(property of complexPack (C3))} \\ &\leq \gamma\alpha \left(\text{fsopt}(\tilde{I})/\beta + 2b\mu\varepsilon \text{opt}(I) \right) + O(1/\varepsilon^2) && \text{(by Lemma 7)} \\ &\leq \gamma\alpha (\rho/\beta + 2b\mu\varepsilon) \text{opt}(I) + O(1/\varepsilon^2) \end{aligned}$$

Here, the last inequality follows from the structural theorem (C2), which says that $\exists(\tilde{I}, D) \in \text{round}(I)$ such that $\text{fsopt}(\tilde{I}) \leq \rho \text{opt}(I) + O(1)$. Hence, the total number of bins is at most

$$|J_{\text{LP}}| + |J_D| + |J| \leq \left((\ln \beta)\mu + \frac{\gamma\alpha\rho}{\beta} + 2b(d_v + 1 + \gamma\alpha\mu)\varepsilon \right) \text{opt}(I) + O(1/\varepsilon^2). \quad \blacktriangleleft$$

C.1 Error in Previous R&A Framework

Here we describe a minor error in the R&A framework of [35], and how it can be fixed.

We define (\tilde{I}, D) as an output of `round`(I) and for the residual instance S , we define \tilde{S} as the corresponding rounded items of $S - D$. Our proof of Lemma 7 relies on the fact that for any subset of rounded items, the span reduces by a factor of at least β if we restrict our attention to the residual instance. Formally, this means that for any $\tilde{X} \subseteq \tilde{I}$, we have

$$\mathbb{E}(\text{span}(\tilde{X} \cap \tilde{S})) = \sum_{i \in \tilde{X}} \text{span}(i) \Pr(i \in \tilde{S}) \leq \text{span}(\tilde{X})/\beta.$$

The equality follows from linearity of expectation and the fact that $\text{span}(i)$ is deterministic, i.e., it doesn't depend on the randomness used in the randomized rounding of the configuration LP. This is because `round` is not given any information about what S is. The inequality follows from Lemma 16, which says that $\Pr(i \in S) \leq 1/\beta$.

The R&A framework of [35] used similar techniques in their analysis. In their algorithm, however, they round items differently. Specifically, they define a subroutine `round` and define $\tilde{I} := \text{round}(I)$ and $\tilde{S} := \text{round}(S)$. They, too, claim that for any subset of rounded items, the span reduces by a factor of at least β if we restrict our attention to the residual instance. While their claim is correct for input-agnostic rounding (where items are rounded up to some constant size collection values chosen independent of the problem instance), the claim is incorrect for input-sensitive rounding (where the values are chosen based on the specific problem instance). So the claim is incorrect if `round` is not deterministic, as then an item can be rounded differently depending on different residual instances.

In fact, they use their R&A framework with the algorithm of Jansen and Prädél [27], which uses *linear grouping* (along with some other techniques) for rounding. Linear grouping rounds items in an *input-sensitive* way, i.e., the rounding of each item depends on the sizes of items in S which is a random subset.

Black Box Absolute Reconstruction for Sums of Powers of Linear Forms

Pascal Koiran ✉

Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Subhayan Saha ✉

Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Abstract

We study the decomposition of multivariate polynomials as sums of powers of linear forms. We give a randomized algorithm for the following problem: If a homogeneous polynomial $f \in K[x_1, \dots, x_n]$ (where $K \subseteq \mathbb{C}$) of degree d is given as a blackbox, decide whether it can be written as a linear combination of d -th powers of linearly independent complex linear forms. The main novel features of the algorithm are:

- For $d = 3$, we improve by a factor of n on the running time from the algorithm in [21]. The price to be paid for this improvement is that the algorithm now has two-sided error.
- For $d > 3$, we provide the first randomized blackbox algorithm for this problem that runs in time $\text{poly}(n, d)$ (in an algebraic model where only arithmetic operations and equality tests are allowed). Previous algorithms for this problem [17] as well as most of the existing reconstruction algorithms for other classes appeal to a polynomial factorization subroutine. This requires extraction of complex polynomial roots at unit cost and in standard models such as the unit-cost RAM or the Turing machine this approach does not yield polynomial time algorithms.
- For $d > 3$, when f has rational coefficients (i.e. $K = \mathbb{Q}$), the running time of the blackbox algorithm is polynomial in n, d and the maximal bit size of any coefficient of f . This yields the first algorithm for this problem over \mathbb{C} with polynomial running time in the bit model of computation.

These results are true even when we replace \mathbb{C} by \mathbb{R} . We view the problem as a tensor decomposition problem and use linear algebraic methods such as checking the simultaneous diagonalisability of the slices of a tensor. The number of such slices is exponential in d . But surprisingly, we show that after a random change of variables, computing just 3 special slices is enough. We also show that our approach can be extended to the computation of the actual decomposition. In forthcoming work we plan to extend these results to overcomplete decompositions, i.e., decompositions in more than n powers of linear forms.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Computing methodologies → Algebraic algorithms; Computing methodologies → Linear algebra algorithms

Keywords and phrases reconstruction algorithms, tensor decomposition, sums of powers of linear forms, simultaneous diagonalisation, algebraic algorithm, black box

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.24

Related Version *Full Version*: <https://arxiv.org/abs/2110.05305>

1 Introduction

Lower bounds and polynomial identity testing are two fundamental problems about arithmetic circuits. In this paper we consider another fundamental problem: arithmetic circuit reconstruction. For an input polynomial f , typically given by a black box, the goal is to find the smallest circuit computing f within some class \mathcal{C} of arithmetic circuits. This problem can be divided in two subproblems: a decision problem (can f be computed by a circuit of size s from the class \mathcal{C} ?) and the reconstruction problem proper (the actual construction of the smallest circuit for f). In this paper we are interested in *absolute reconstruction*,



© Pascal Koiran and Subhayan Saha;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 24; pp. 24:1–24:17



Leibniz International Proceedings in Informatics

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

namely, in the case where \mathcal{C} is a class of circuits over the field of complex numbers. The name is borrowed from *absolute factorization*, a well-studied problem in computer algebra (see e.g. [9, 10, 11, 24]). Most of the existing reconstruction algorithms appeal to a polynomial factorization subroutine, see e.g. [12, 13, 16, 17, 18, 19, 26]. This typically yields polynomial time algorithms over finite fields or the field of rational numbers. However, in standard models of computation such as the unit-cost RAM or the Turing machine this approach does *not* yield polynomial time algorithms for absolute reconstruction. This is true even for the decision version of this problem. In the Turing machine model, the difficulty is as follows. We are given an input polynomial f , say with rational coefficients, and want to decide if there is a small circuit $C \in \mathcal{C}$ for f , where C may have complex coefficients. After applying a polynomial factorization subroutine, a reconstruction algorithm will manipulate polynomials with coefficients in a field extension of \mathbb{Q} . If this extension is of exponential degree, the remainder of the algorithm will not run in polynomial time. This point is explained in more detail in [21] on the example of a reconstruction algorithm due to Neeraj Kayal [17]. One way out of this difficulty is to work in a model where polynomial roots can be extracted at unit cost, as suggested in a footnote of [14]. We will work instead in more standard models, namely, the Turing machine model or the unit-cost RAM over \mathbb{C} with arithmetic operations only (an appropriate formalization is provided by the Blum-Shub-Smale model of computation [5, 6]). Before presenting our results, we present the class of circuits studied in this paper.

1.1 Sums of powers of linear forms

Let $f(x_1, \dots, x_n)$ be a homogeneous polynomial of degree d . In this paper we study decompositions of the type:

$$f(x_1, \dots, x_n) = \sum_{i=1}^r l_i(x_1, \dots, x_n)^d \quad (1)$$

where the l_i are linear forms. Such a decomposition is sometimes called a Waring decomposition, or a symmetric tensor decomposition. The smallest possible value of r is the symmetric tensor rank of f , and it is NP-hard to compute already for $d = 3$ [25]. One can nevertheless obtain polynomial time algorithms by restricting to a constant value of r [3]. In this paper we assume instead that the linear forms l_i are linearly independent (hence $r \leq n$). This setting was already studied by Kayal [17]. It turns out that such a decomposition is unique when it exists, up to a permutation of the l_i and multiplications by d -th roots of unity. This follows for instance from Kruskal's uniqueness theorem. For a more elementary proof, see [17, Corollary 5.1] and [21, Section 3.1].

Under this assumption of linear independence, the case $r = n$ is of particular interest. In this case, f is *equivalent* to the sum of d -th powers polynomial

$$P_d(x) = x_1^d + x_2^d + \dots + x_n^d \quad (2)$$

in the sense that $f(x) = P_d(Ax)$ where A is invertible. A test of equivalence to P_d was provided in [17]. The resulting algorithm provably runs in polynomial time over the field of rational numbers, but this is not the case over \mathbb{C} due to the appeal to polynomial factorization. The first equivalence test to P_d running in polynomial time over the field of complex numbers was given in [21] for $d = 3$. We will extend this result to arbitrary degree in this paper. In the general case $r \leq n$ we can first compute the number of essential variables of f [8, 17]. Then we can do a change of variables to obtain a polynomial depending only on its first r variables [17, Theorem 4.1], and conclude with a test of equivalence to P_r (see [21, Proposition 44] for details).

Equivalence and reconstruction algorithms over \mathbb{Q} are number-theoretic in nature in the sense that their behavior is highly sensitive to number-theoretic properties of the coefficients of the input polynomial. This point is clearly illustrated by an example from [21]:

► **Example 1.** Consider the rational polynomial

$$f(x_1, x_2) = (x_1 + \sqrt{2}x_2)^3 + (x_1 - \sqrt{2}x_2)^3 = 2x_1^3 + 12x_1x_2^2.$$

This polynomial is equivalent to $P_3(x_1, x_2) = x_1^3 + x_2^3$ over \mathbb{R} and \mathbb{C} but not over \mathbb{Q} .

By contrast, equivalence and reconstruction algorithms over \mathbb{R} and \mathbb{C} are of a more geometric nature.

1.2 Connection to Tensor Decomposition

Using the relation between tensors and polynomials, we can see that a homogeneous degree- d polynomial $f \in \mathbb{K}[x_1, \dots, x_n]$ can be written as a sum of d -th powers of linear forms over \mathbb{K} if and only if there exist $v_i \in \mathbb{K}$ such that the corresponding symmetric tensor T_f can be decomposed as $T_f = \sum_i v_i^{\otimes d}$. This is often referred to as the tensor decomposition problem for the given tensor T .

Most tensor decompositions algorithms are numerical such as the ALS method [22] (which lacks a good complexity analysis), tensor power iteration [1] (for orthogonal tensor decomposition) or Jennrich's algorithm [15, 23] for ordinary tensors. A self-contained bit-complexity analysis of Jennrich's algorithm can be found in [4]. Unlike the above algorithm from [21], these numerical algorithms do not provide any decision procedure. The algebraic algorithm from [21] seems closest in spirit to Jennrich's: they both rely on simultaneous diagonalization and on linear independence assumptions on the vectors involved in the tensor decomposition. Algorithms for symmetric tensor decomposition can be found in the algebraic literature, see e.g. [2, 7]. These two papers do not provide any complexity analysis.

1.3 Results and methods

Our main contributions are as follows. Recall that P_d is the sum of d -th powers polynomials (2), and let us assume that the input $f \in \mathbb{C}[x_1, \dots, x_n]$ is a homogeneous polynomial of degree d .

- (i) For $d = 3$, we improve by a factor of n on the running time of the test of equivalence to P_3 from [21]. The price to be paid for this improvement is that the algorithm now has two-sided error. The algorithm for the $d = 3$ case and a simpler analysis can be found in the full paper [20].
- (ii) For $d > 3$, we provide the first blackbox algorithm for equivalence to P_d with running time polynomial in n and d (more specifically, $O(n^2d)$ calls to the blackbox and $O(n^2d \log^2(d) \log \log(d) + n^{\omega+1})$ arithmetic operations) where ω is the exponent of matrix multiplication, in an algebraic model where only arithmetic operations and equality tests are allowed (i.e., computation of polynomial roots is *not allowed*).
- (iii) For $d > 3$, when f has rational coefficients this blackbox algorithm runs in polynomial time in the bit model of computation. More precisely, the running time is polynomial in n, d and the maximal bit size of any coefficient of f . This yields the first test of equivalence to P_d over \mathbb{C} with polynomial running time in the bit model of computation.

As outlined in Section 1.1, these results have application to decomposition into sums of powers of linearly independent linear forms over \mathbb{C} . Namely, we can decide whether the input polynomial admits such a decomposition, and if it does we can compute the number of terms

r in such a decomposition. The resulting algorithm runs in polynomial time in the algebraic model of computation, as in item (ii) above; when the input has rational coefficients it runs in polynomial time in the bit model of computation, as in (iii) (refer to Appendix B in the full paper [20] for a detailed complexity analysis). This is the first algorithm with these properties. It can be viewed as an algebraic, high order, black box version of Jennrich’s algorithm.

Using the relation to tensor decomposition problem mentioned in Section 1.2, if an order d -tensor $T \in K^{n \times \dots \times n}$ is given as a blackbox, we give an algorithm that runs in time $\text{poly}(n, d)$ to check if there exist linearly independent vectors $v_i \in \mathbb{K}^n$ such that $T = \sum_{i=1}^t \alpha_i v_i^{\otimes d}$ for some $t \leq n$. Note here that $K \subseteq \mathbb{C}$ and $\mathbb{K} = \mathbb{C}$ or \mathbb{R} . This can be found in more detail in Section 4 of the full paper [20].

Finally, in Section 5 of the full paper [20], we show that our linear algebraic approach can be extended to the computation of the actual decomposition in a model where we allow the computation of polynomial roots. We therefore obtain an alternative to the algorithm from [17] for this problem. That algorithm relies on multivariate polynomial factorization, whereas our algorithm relies on matrix diagonalization.

Real versus complex field. For $\mathbb{K} = \mathbb{R}$ and even degree there is obviously a difference between sums of d -th powers of linear forms and linear combinations of d -th powers. In this paper we wish to allow arbitrary linear combinations. For this reason, in the treatment of the high order case ($d > 3$) we are not interested in equivalence to P_d only. Instead, we would like to know whether the input is equivalent to some polynomial of the form $\sum_{i=1}^n \alpha_i x_i^d$ with $\alpha_i \neq 0$ for all i . We denote by \mathcal{P}_d this class of polynomials (one could even assume that $\alpha_i = \pm 1$ for all i). At first reading, there is no harm in assuming that $\mathbb{K} = \mathbb{C}$. In this case, one can assume without loss of generality that $\alpha_i = 1$ for all i . For $\mathbb{K} = \mathbb{R}$, having to deal with the whole of \mathcal{P}_d slightly complicates notations, but the proofs are not significantly more complicated than for $\mathbb{K} = \mathbb{C}$. For this reason, in all of our results we give a unified treatment of the two cases $\mathbb{K} = \mathbb{C}$ and $\mathbb{K} = \mathbb{R}$.

Methods. We associate to a homogeneous polynomial of degree d the (unique) symmetric tensor T of order d such that

$$f(x_1, \dots, x_n) = \sum_{i_1, \dots, i_d=1}^n T_{i_1 \dots i_d} x_{i_1} x_{i_2} \dots x_{i_d}.$$

We recall that T is said to be symmetric if it is invariant under all $d!$ permutations of its indices. A *slice* of T (or by abuse of language, a slice of f) is a matrix of size n obtained by fixing the values of $d - 2$ indices. In Section 2.1 we give a characterization of equivalence to P_d :

► **Theorem 2.** *A degree d homogeneous polynomial $f \in \mathbb{C}[x_1, \dots, x_n]$ is equivalent to $P_d = \sum_{i=1}^n x_i^d$ if and only if its slices span a nonsingular matrix space and the slices are simultaneously diagonalizable by congruence, i.e., there exists an invertible matrix $Q \in M_n(\mathbb{C})$ such that for every slice S of f , the matrix $Q^T S Q$ is diagonal.*

This characterization is satisfactory from a purely structural point of view, but not from an algorithmic point of view because a tensor of order d has $\frac{d(d-1)}{2} n^{d-2}$ slices. The tensors encountered in this paper are all symmetric since they originate from homogeneous polynomials. Taking the symmetry constraints into consideration reduces the number of distinct slices to $\binom{n+d-3}{d-2}$ at most: this is the number of multisets of size $d - 2$ in a set of n elements,

or equivalently the number of monomials of degree $d - 2$ in the variables x_1, \dots, x_n . This number remains much too large to reach our goal of a complexity polynomial in n and d . This problem has a surprisingly simple solution: our equivalence algorithm (Algorithm 1 below) needs to work with 3 slices only! This is true already for $d = 3$, and is the reason why we can save a factor of n compared to the algorithm of [21]. More detail on the degree 3 case can be found in Section 2 of the full paper [20].

■ **Algorithm 1** Randomized algorithm to check polynomial equivalence to an element of \mathcal{P}_d .

Input: A degree- d homogeneous polynomial f
 Let $R \in M_n(\mathbb{K})$ be a matrix such that its entries r_{ij} are picked uniformly and independently at random from a finite set S , and set $h(x) = f(Rx)$.
 Let $\{T_{i_1 \dots i_{d-2}}\}_{i_1 \dots i_{d-2} \in [n]}$ be the slices of h .
 We compute the slices T_1, T_2, T_3 , where T_i refers to the slice $T_{i \dots i}$.
if T_1 *is singular* **then**
 | reject
else
 | compute $T'_1 = (T_1)^{-1}$
 | **if** $T'_1 T_2$ and $T'_1 T_3$ commute and $T'_1 T_2$ is diagonalisable over \mathbb{K} **then**
 | | accept
 | **else**
 | | reject
 | **end**
end

More precisely, the following test forms the heart of the algorithm: check that $T_1^{-1} T_2$ is diagonalizable, and commutes with $T_1^{-1} T_3$ where T_i refers to the slice $T_{i \dots i}$. It may be surprising at first sight that we can work with 3 slices only of a tensor with $\binom{n+d-3}{d-2}$ slices. To give some plausibility to this claim, note that T_1, T_2, T_3 are not slices of the input f , but slices of the polynomial $h(x) = f(Rx)$ obtained by a random change of variables. As a result, each slice of h contains some information on *all* of the slices of f .

Our algorithms are therefore quite simple but their analysis is quite non-trivial. In fact, analysing the case of “negative” inputs, i.e. input polynomials that are not equivalent to any polynomial in \mathcal{P}_d , forms the bulk of this paper. For $d > 3$, the notion of “weak-singularity” of matrices (Definition 14) has been introduced which along with the notions of “commutativity property” and “diagonalisability property” helps us to give us another equivalent criterion for testing equivalence to a polynomial in \mathcal{P}_d in Theorem 15. Finally, the crucial part of the proof (for $d > 3$, and already for $d = 3$) is to show that testing commutativity of two matrices and diagonalisability of one matrix is enough for testing these properties for any “symmetric family of symmetric matrices” (refer to Definition 19) with high probability.

Note here that an arbitrary slice of the polynomial is hard to compute, when the polynomial is given as blackbox (because that requires computing arbitrary degree- d partial derivatives using the blackbox). Hence, this particular choice of slices is crucial because they can be computed in polynomial time.

1.4 Notations

We work in a field \mathbb{K} which may be the field of real numbers or the field of complex numbers. Some of our intermediate results apply to other fields as well. We denote by $\mathbb{K}[x_1, \dots, x_n]_d$ the space of homogeneous polynomials of degree d in n variables with coefficients in \mathbb{K} . A

homogeneous polynomial of degree d is also called a *degree- d form*. We denote by P_d the polynomial $\sum_{i=1}^n x_i^d$, and we say that a degree d form $f(x_1, \dots, x_n)$ is equivalent to a sum of d -th powers if it is equivalent to P_d , i.e., if $f(x) = P_d(Ax)$ for some invertible matrix A . More generally, we denote by \mathcal{P}_d the set of polynomials of the form $\sum_{i=1}^n \alpha_i x_i^d$ with $\alpha_i \neq 0$ for all i . As explained in Section 1.3, for $\mathbb{K} = \mathbb{R}$ we are not only interested in equivalence to P_d : we would like to know whether the input is equivalent to one of the elements of \mathcal{P}_d .

We denote by $M_n(\mathbb{K})$ the space of square matrices of size n with entries from \mathbb{K} . We denote by ω a feasible exponent for matrix multiplication, i.e., we assume that two matrices of $M_n(\mathbb{K})$ can be multiplied with $O(n^\omega)$ arithmetic operations in \mathbb{K} .

Throughout the paper, we will choose the entries r_{ij} of a matrix R independently and uniformly at random from a finite set $S \subset \mathbb{K}$. When we calculate the probability of some event E over the random choice of the r_{ij} , by abuse of notation instead of $\Pr_{r_{11}, \dots, r_{nn} \in S}[E]$ we simply write $\Pr_{R \in S}[E]$.

2 Equivalence to a linear combination of d -th powers

We can associate to a symmetric tensor T of order d the homogeneous polynomial

$$f(x_1, \dots, x_n) = \sum_{i_1, \dots, i_d \in [n]} T_{i_1 \dots i_d} x_{i_1} \dots x_{i_d}.$$

This correspondence is bijective, and the symmetric tensor associated to a homogeneous polynomial f can be obtained from the relation

$$T_{i_1 \dots i_d} = \frac{1}{d!} \frac{\partial^d f}{\partial x_{i_1} \dots \partial x_{i_d}}.$$

The (i_1, \dots, i_{d-2}) -th slice of T is the symmetric matrix $T_{i_1 \dots i_{d-2}}$ with entries $(T_{i_1 \dots i_{d-2}})_{i_{d-1}, i_d} = T_{i_1 \dots i_d}$. Recall from Section 1.4 that we denote by \mathcal{P}_d the set of polynomials of the form $\sum_{i=1}^n \alpha_i x_i^d$ with $\alpha_i \neq 0$ for all $i \in [n]$. In this section we analyze Algorithm 1. Recall from the introduction that this is a polynomial time algorithm for checking whether an input degree d form in n variables f is equivalent to some polynomial in \mathcal{P}_d . This means that $f(x) = P_d(Ax)$ for some $P_d \in \mathcal{P}_d$ and some invertible matrix A .

We prove the surprising fact that even when the input polynomial has degree d , checking commutativity of 2 matrices and the diagonalisability of one matrix is enough to check equivalence to a polynomial of \mathcal{P}_d . Let $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of $h(x) = f(Rx)$. Recall that T_i denotes the slice $T_{i \dots i}$. Algorithm 1 checks if $T_1' T_2$ commutes with $T_1' T_3$ and if $T_1' T_3$ is diagonalisable.

Interestingly though, arbitrary slices of a degree- d polynomial are hard to compute. These particular slices are special because they can be computed using a small number of calls to the blackbox and in small number of arithmetic operations (due to the fact that they are essentially repeated partial derivatives with respect to a single variable). Hence, they help us give a polynomial time algorithm. More precisely, we show that if the polynomial is given as a blackbox, the algorithm requires only $O(n^2 d)$ calls to the blackbox and $O(n^2 M(d) \log d + n^{\omega+1})$ many arithmetic operations. We do a detailed complexity analysis of this algorithm in Appendix B in the full paper [20].

The remainder of this section is devoted to a correctness proof for Algorithm 1, including an analysis of the probability of error. Our main result about this algorithm is as follows:

► **Theorem 3.** *If an input $f \in \mathbb{F}[x_1, \dots, x_n]_d$ is not equivalent to some polynomial $P_d \in \mathcal{P}_d$, then f is rejected by the algorithm with high probability over the choice of the random matrix R . More precisely, if the entries $r_{i,j}$ of R are chosen uniformly and independently at random*

from a finite set $S \subseteq \mathbb{K}$, then the input will be rejected with probability $\geq 1 - \frac{2(d-2)}{|S|}$. Conversely, if f is equivalent to some polynomial $P_d \in \mathcal{P}_d$, then f will be accepted with high probability over the choice of the random matrix R . More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be accepted with probability $\geq 1 - \frac{n(d-1)}{|S|}$.

In Section 2.2, we give a proof of the second part of theorem, i.e., we analyze the behavior of Algorithm 1 on the polynomials that can be decomposed as a sum of d th powers of linearly independent linear forms (which we refer to as the positive inputs). We give a characterization of positive inputs in terms of the subspace of matrices spanned by their slices. Namely, we show that these subspaces must not be “weakly singular”, and must satisfy the commutativity and diagonalizability properties. If a polynomial is not equivalent to some polynomial in \mathcal{P}_d , this can happen in several ways depending on which property fails. We analyze the failure of commutativity in Section 2.3.1 and failure of diagonalisability in Section 2.3.2. Then we collect everything together and prove the first part of the theorem in Section 2.3.3.

2.1 Characterisation of equivalence to \mathcal{P}_d

First, we show how the slices of a degree- d form evolve under a linear change of variables. Towards the proof of Theorem 2 we need some results from [21], which we recall in this section. We also give a converse of this in Theorem 8. First, let us recall how the slices of a polynomial evolve under a linear change of variables.

► **Theorem 4.** *Let g be a degree- d form with slices $\{S_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ and let $f(x) = g(Ax)$. Then the slices $T_{i_1 \dots i_{d-2}}$ of f , are given by $T_{i_1 \dots i_{d-2}} = A^T D_{i_1 \dots i_{d-2}} A$ where $D_{i_1 \dots i_{d-2}} = \sum_{j_1 \dots j_{d-2} \in [n]} a_{j_1 i_1} \dots a_{j_{d-2} i_{d-2}} S_{j_1 \dots j_{d-2}}$ and $a_{i,j}$ are the entries of A . If $g = \sum_{i=1}^n \alpha_i x_i^d$, we have $D_{i_1 \dots i_{d-2}} = \text{diag}(\alpha_1 (\prod_{m=1}^{d-2} a_{1, i_m}), \dots, \alpha_n (\prod_{m=1}^{d-2} a_{n, i_m}))$.*

Proof. By definition of the slices of a polynomial,

$$S_{i_1 \dots i_{d-2}} = \frac{1}{d!} H_{\frac{\partial^{d-2} g}{\partial x_{i_1} \dots \partial x_{i_{d-2}}}}(x) \text{ and } T_{i_1 \dots i_{d-2}} = \frac{1}{d!} H_{\frac{\partial^{d-2} f}{\partial x_{i_1} \dots \partial x_{i_{d-2}}}}(x)$$

where $H_f(x)$ is the Hessian matrix of f at point x . Since $f(x) = g(Ax)$, by differentiating d times, we get that $\frac{\partial^d f}{\partial x_{i_1} \dots \partial x_{i_d}}(x) = \sum_{j_1 \dots j_d \in [n]} a_{j_1 i_1} \dots a_{j_d i_d} \frac{\partial^d g}{\partial x_{j_1} \dots \partial x_{j_d}}(Ax)$. Putting these equations in matrix form, and using the fact that $\frac{\partial^d g}{\partial x_{j_1} \dots \partial x_{j_d}}(Ax) = \frac{\partial^d g}{\partial x_{j_1} \dots \partial x_{j_d}}(x)$ we get the desired result. ◀

Instead of diagonalisation by congruence, it is convenient to work with the more familiar notion of diagonalisation by similarity, where an invertible matrix A acts by $S \mapsto A^{-1}SA$ instead of $A^T SA$. We collect the necessary material in the remainder of this section (and we refer to diagonalisation by similarity simply as *diagonalisation*).

The two following properties play a fundamental role throughout the paper.

► **Definition 5.** *Let \mathcal{V} be a non-singular space of matrices.*

- We say that \mathcal{V} satisfies the **Commutativity Property** if there exists an invertible matrix $A \in \mathcal{V}$ such that $A^{-1}\mathcal{V}$ is a commuting subspace, i.e., $PQ = QP$ for any two matrices $P, Q \in A^{-1}\mathcal{V}$.
- We say that \mathcal{V} satisfies the **Diagonalisability Property** if there exists an invertible matrix $B \in \mathcal{V}$ such that all the matrices in the space $B^{-1}\mathcal{V}$ are diagonalisable.

The next result can be found in [21, Section 2.2].

► **Theorem 6.** *Let \mathcal{V} be a non-singular subspace of matrices of $M_n(\mathbb{K})$. The following properties are equivalent.*

- \mathcal{V} satisfies the commutativity property.
- For all non-singular matrices $A \in \mathcal{V}$, $A^{-1}\mathcal{V}$ is a commuting subspace.

► **Remark 7.** Let \mathcal{V} be a non-singular subspace of matrices which satisfies the commutativity and diagonalisability properties. There exists an invertible matrix $B \in \mathcal{V}$ and an invertible matrix R which diagonalizes simultaneously all of $B^{-1}\mathcal{V}$ (i.e., $R^{-1}MR$ is diagonal for all $M \in B^{-1}\mathcal{V}$).

Proof. Pick an invertible matrix $B \in \mathcal{V}$ such that $\mathcal{W} = B^{-1}\mathcal{V}$ is a space of diagonalizable matrices. By Theorem 6, \mathcal{W} is a commuting subspace. It is well known that a finite collection of matrices is simultaneously diagonalisable if and only if they commute, and each matrix in the collection is diagonalisable. We conclude by applying this result to a basis of \mathcal{W} (any matrix R which diagonalises a basis will diagonalise all of \mathcal{W}). ◀

We now give an analogue of Theorem 6 for the diagonalisability property.

► **Theorem 8.** *Let \mathcal{V} be a non-singular subspace of matrices which satisfies the commutativity property. The following properties are equivalent:*

- \mathcal{V} satisfies the diagonalisability property.
- For all non-singular matrices $A \in \mathcal{V}$, the matrices in $A^{-1}\mathcal{V}$ are simultaneously diagonalisable.

Proof. Suppose that \mathcal{V} satisfies the diagonalisability property. By the previous remark, we already know that there exists *some* invertible matrix $B \in \mathcal{V}$ such that the matrices in $B^{-1}\mathcal{V}$ are simultaneously diagonalisable by an invertible matrix R . We need to establish the same property for an arbitrary invertible matrix $A \in \mathcal{V}$. For any $M \in \mathcal{V}$, $A^{-1}M = (B^{-1}A)^{-1}(B^{-1}M)$. Hence $A^{-1}M$ is diagonalised by R since this matrix diagonalises both matrices $B^{-1}A$ and $B^{-1}M$. Since R is independent of the choice of $M \in \mathcal{V}$, we have shown that the matrices in $A^{-1}\mathcal{V}$ are simultaneously diagonalisable. ◀

The importance of the commutativity and diagonalisability properties stems from the fact that they provide a characterization of simultaneous diagonalisation by congruence, which in turn provides a characterization of equivalence to some polynomial in \mathcal{P}_d :

► **Theorem 9.** *Let $A_1, \dots, A_k \in M_n(\mathbb{K})$ and assume that the subspace \mathcal{V} spanned by these matrices is non-singular. There are diagonal matrices Λ_i and a non-singular matrix $R \in M_n(\mathbb{K})$ such that $A_i = R\Lambda_iR^T$ for all $i \in [k]$ if and only if \mathcal{V} satisfies the Commutativity property and the Diagonalisability property.*

For a proof, see [21, Section 2.2] for $\mathbb{K} = \mathbb{C}$ and [21, Section 2.3] for $\mathbb{K} = \mathbb{R}$.

The next lemma uses Theorem 4 to reveal some crucial properties about the subspace spanned by the slices of any degree- d form which is equivalent to some $g \in \mathcal{P}_d$.

► **Lemma 10.** *Let $f(x_1, \dots, x_n)$ and $g(x_1, \dots, x_n)$ be two forms of degree d such that $f(x) = g(Ax)$ for some non-singular matrix A .*

1. If \mathcal{U} and \mathcal{V} denote the subspaces of $M_n(\mathbb{K})$ spanned respectively by the slices of f and g , we have $\mathcal{U} = A^T\mathcal{V}A$.
2. \mathcal{V} is non-singular iff \mathcal{U} is non-singular.
3. In particular, for $g \in \mathcal{P}_d$ the subspace \mathcal{V} is the space of diagonal matrices and \mathcal{U} is a non-singular subspace, i.e., it is not made of singular matrices only.

Proof. Theorem 4 shows that $\mathcal{U} \subseteq A^T \mathcal{V} A$. Now since, $g(x) = f(A^{-1}x)$, same argument shows that $\mathcal{V} \subseteq A^{-T} \mathcal{U} A^{-1}$. This gives us that $\mathcal{U} = A^T \mathcal{V} A$.

For the second part of the lemma, let us assume that \mathcal{V} is non-singular and $M_{\mathcal{U}}$ be an arbitrary matrix in \mathcal{U} . Using the previous part of the lemma, we know that there exists $M_{\mathcal{V}} \in \mathcal{V}$ such that $M_{\mathcal{U}} = A^T M_{\mathcal{V}} A$. Since \mathcal{V} is non-singular, $\det(M_{\mathcal{V}}) \neq 0$. Taking determinant on both sides, we get that $\det(M_{\mathcal{U}}) = \det(A)^2 \det(M_{\mathcal{V}}) \neq 0$ (since A is invertible, $\det(A) \neq 0$). For the converse, assume that \mathcal{U} is non-singular. Following a similar proof, it can be shown that $\det(M_{\mathcal{U}}) \neq 0$.

For the third part of the lemma, let $\{S_{i_1 \dots i_{d-2}}\}_{i_1 \dots i_{d-2} \in [n]}$ be the slices of g . If $g = \sum_{i \in [n]} \alpha_i x_i^d$, such that $\alpha_i \neq 0$ for all i , S_i has α_i in the (i, i) -th position and 0 everywhere. Also, $S_{i_1, \dots, i_{d-2}} = 0$, when the i_k 's are not equal. Hence, \mathcal{V} is the space of all diagonal matrices. Hence \mathcal{V} is a non-singular space. Using the previous part of the lemma, we get that \mathcal{U} is a non-singular space as well. ◀

The next lemma is effectively a converse of the second part of Lemma 10. It shows that if the slices of f are diagonal matrices, then the fact that they effectively originate from a symmetric tensor forces them to be extremely special.

► **Lemma 11.** *Let $f \in \mathbb{K}[x_1, \dots, x_n]_d$ be a degree- d form. If the slices of f are diagonal matrices, then $f = \sum_{i \in [n]} \alpha_i x_i^d$ for some $\alpha_1, \dots, \alpha_n \in \mathbb{K}$.*

Proof. Let $T_{i_1, \dots, i_{d-2}}$ be the slices of f . Let $I = \{(i_{\sigma(1)}, \dots, i_{\sigma(d)}) \mid \sigma \in S_d\}$. Now since they are slices of a polynomial, we know that

$$(T_{i_1 \dots i_{d-2}})_{i_{d-1}, i_d} = (T_{i_{\sigma(1)}, \dots, i_{\sigma(d-2)}})_{i_{\sigma(d-1)}, i_{\sigma(d)}}. \quad (3)$$

We want to show that $T_{i_1, \dots, i_d} \neq 0$ only if $i_1 = i_2 = \dots = i_d$. Using (3), it is sufficient to show that $(T_{i_1, \dots, i_{d-2}})_{i_{d-1}, i_d} \neq 0$ only if $i_{d-1} = i_d$. This is true since $T_{i_1, \dots, i_{d-2}}$ are diagonal matrices. This gives us that $f = \sum_{i \in [n]} \alpha_i x_i^d$. ◀

Now we are finally ready to prove a theorem that characterizes exactly the set of degree- d homogeneous polynomials which are equivalent to some $g \in \mathcal{P}_d$. This already appears as Theorem 2 in the introduction. We restate it now for the reader's convenience.

► **Theorem 12.** *A degree d form $f \in \mathbb{K}[x_1, \dots, x_n]$ is equivalent to some polynomial $P_d \in \mathcal{P}_d$ if and only if its slices $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ span a non-singular matrix space and the slices are simultaneously diagonalisable by congruence, i.e., there exists an invertible matrix $Q \in M_n(\mathbb{K})$ such that the matrices $Q^T T_{i_1 \dots i_{d-2}} Q$ are diagonal for all $i_1, \dots, i_{d-2} \in [n]$.*

Proof. Let \mathcal{U} be the space spanned by $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$. If f is equivalent to P_d , Theorem 4 shows that the slices of f are simultaneously diagonalisable by congruence and Lemma 10 shows that \mathcal{U} is non-singular.

Let us show the converse. Since the slices $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ are simultaneously diagonalisable, there are diagonal matrices $\Lambda_{i_1 \dots i_{d-2}}$ and a non-singular matrix $R \in M_n(\mathbb{K})$ such that $T_{i_1 \dots i_{d-2}} = R \Lambda_{i_1 \dots i_{d-2}} R^T$ for all $i_1, \dots, i_{d-2} \in [n]$. So now we consider $g(x) = f(R^{-T}x)$. Let $\{S_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of g . Using Theorem 4, we get that $S_{i_1 \dots i_{d-2}} = (R^{-1}) (\sum_{j_1 \dots j_{d-2} \in [n]} r_{j_1 i_1} \dots r_{j_{d-2} i_{d-2}} R \Lambda_{j_1 \dots j_{d-2}} R^T) R^{-T} = \sum_{j_1 \dots j_{d-2} \in [n]} r_{j_1 i_1} \dots r_{j_{d-2} i_{d-2}} \Lambda_{j_1 \dots j_{d-2}}$. This implies that $S_{j_1 \dots j_{d-2}}$ are also diagonal matrices. By Lemma 11, $g = \sum_{i \in [n]} \alpha_i x_i^d$. It therefore remains to be shown that $\alpha_i \neq 0$, for all $i \in [n]$. Let \mathcal{V} be the subspace spanned by the slices of g and the slices of f span a non-singular matrix space \mathcal{U} . Since, \mathcal{U} is a non-singular subspace of matrices, using part (2) of Lemma 10, we get that \mathcal{V} is a non-singular subspace of matrices.

24:10 Black Box Absolute Reconstruction for Sums of Powers of Linear Forms

But if some α_i vanishes, for all $A \in \mathcal{V}$, $A_{\bar{i}} = 0$. Hence \mathcal{V} is a singular subspace, which is a contradiction. This gives us that $g = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$ for all i . Hence, $g \in \mathcal{P}_d$ and f is equivalent to g . \blacktriangleleft

► **Theorem 13.** *Let $f \in \mathbb{K}[x_1, \dots, x_n]$ be a degree- d form. f is equivalent to some polynomial $P_d \in \mathcal{P}_d$ iff the subspace \mathcal{V} spanned by its slices $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ is a non-singular subspace and \mathcal{V} satisfies the Commutativity Property and the Diagonalisability Property.*

Proof. This follows from Theorem 12 and Theorem 9 for $k = n^{d-2}$ to get the result. \blacktriangleleft

Notice here that the notion of weak-singularity is entirely dependent on the generating set of matrices. So it is more of a property of the generating set. But by abuse of language, we will call the span of the matrices weakly singular. To put it in contrast, usually the notion of singularity is a property of the subspace spanned by the matrices (irrespective of the generating set). It can be further observed that for all $n \geq 2$ and $d \geq 4$, non-singular families of matrices can be easily constructed which are weakly singular!

► **Definition 14** (Weak singularity). *Let \mathcal{V} be the space spanned by matrices $\{S_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$. \mathcal{V} is weakly singular if for all $\alpha = (\alpha_1, \dots, \alpha_n)$, $\det(\sum_{i_1, \dots, i_{d-2} \in [n]} (\prod_{k \in [d-2]} \alpha_{i_k}) S_{i_1 \dots i_{d-2}}) = 0$.*

Notice here that the notion of weak-singularity is entirely dependent on the generating set of matrices. So it is more of a property of the generating set. But by abuse of language, we will call the span of the matrices weakly singular. To put it in contrast, usually the notion of singularity is a property of the subspace spanned by the matrices (irrespective of the generating set).

► **Theorem 15.** *Let $f \in \mathbb{K}[x_1, \dots, x_n]$ be a degree- d form. f is equivalent to some polynomial $P_d \in \mathcal{P}_d$ iff the subspace \mathcal{V} spanned by its slices $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ is not a weakly singular subspace, satisfies the Commutativity Property and the Diagonalisability Property.*

Proof. First we show that if $f = P_d(Ax)$ such that $P_d \in \mathcal{P}_d$ i.e. $P_d(x) = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$ for all $i \in [n]$ and A is invertible, then \mathcal{V} is not a weakly singular subspace, satisfies the commutativity property and the diagonalisability property.

Let $\{S_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of P_d . Then $S_{\bar{i}} = \alpha_i \text{diag}(e_i)$ where e_i is the i -th standard basis vector, and all other slices are 0. From Theorem 4,

$$T_{i_1 \dots i_{d-2}} = A^T D_{i_1 \dots i_{d-2}} A = A^T \left(\sum_{k \in [n]} a_{ki_1} \dots a_{ki_{d-2}} S_{\bar{k}} \right) A.$$

Now we define

$$\begin{aligned} T(\bar{\beta}) &= \sum_{i_1, \dots, i_{d-2} \in [n]} \left(\prod_{k \in [d-2]} \beta_{i_k} \right) T_{i_1 \dots i_{d-2}} \\ &= \sum_{i_1, \dots, i_{d-2} \in [n]} \left(\prod_{k \in [d-2]} \beta_{i_k} \right) A^T \left(\text{diag} \left(\alpha_1 \left(\prod_{m \in [d-2]} a_{1i_m} \right), \dots, \alpha_n \left(\prod_{m \in [d-2]} a_{ni_m} \right) \right) \right) A \\ &= A^T \text{diag} \left(\alpha_1 \left(\sum_{i_1, \dots, i_{d-2} \in [n]} \left(\prod_{k \in [d-2]} \beta_{i_k} a_{1i_k} \right) \right), \dots, \alpha_n \left(\sum_{i_1, \dots, i_{d-2} \in [n]} \left(\prod_{k \in [d-2]} \beta_{i_k} a_{ni_k} \right) \right) \right) A. \end{aligned}$$

Taking determinant on both sides, $\det(T)(\bar{\beta}) = \det(A)^2 \prod_{m=1}^n T_m(\bar{\beta})$ where $T_m(\bar{\beta}) = \alpha_m \left(\sum_{i_1, \dots, i_{d-2} \in [n]} \left(\prod_{k \in [d-2]} \beta_{i_k} a_{mi_k} \right) \right)$. Since, A is invertible, none of its rows are all 0. Hence for all $m_0 \in [n]$, there exists $j_0 \in [n]$, such that $a_{m_0 j_0} \neq 0$. Then $\text{coeff}_{\beta_{j_0}^{d-2}}(T_{m_0}) = a_{m_0 j_0}^{d-2} \neq 0$. Hence $T_{m_0} \neq 0$ for all $m_0 \in [n]$ which implies that $\det(T) \neq 0$. Therefore, there exists $\bar{\beta}^0$ such that $\det(T)(\bar{\beta}^0) \neq 0$. This proves that $\det(\sum_{i_1, \dots, i_{d-2} \in [n]} (\prod_{k \in [d-2]} \beta_{i_k}) T_{i_1 \dots i_{d-2}}) \neq 0$.

Hence, $\mathcal{V} = \text{span}\{T_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ is not weakly singular. Theorem 13 gives us that the subspace spanned by the slices \mathcal{V} satisfies the commutativity property and the diagonalisability property.

For the converse, if \mathcal{V} is not a weakly singular subspace, then it is a non-singular subspace as well. And it satisfies the commutativity property and the diagonalisability property. By Theorem 13, we get that f is equivalent to some polynomial in \mathcal{P}_d . ◀

2.2 Analysis for positive inputs

In this section we analyze the behavior of Algorithm 1 on inputs that are equivalent to some polynomial in \mathcal{P}_d (which we refer to as the positive inputs). We recall here again that by $T_{\bar{1}}$, we denote the slice $T_{11 \dots 1}$.

► **Lemma 16.** *Let $f \in \mathbb{K}[x_1, \dots, x_n]_d$ with slices $\{S_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$, such that the subspace \mathcal{V} spanned by the slices is not weakly singular. Let $h(x) = f(Rx)$ where the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$. Let $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of h . Then $\Pr_{R \in S}[T_{\bar{1}} \text{ is invertible}] \geq 1 - \frac{n(d-1)}{|S|}$.*

Proof. We can obtain the slices $T_{i_1 \dots i_{d-2}}$ of h from the slices $S_{i_1 \dots i_{d-2}}$ of f using Theorem 4. Namely, we have $T_{i_1 \dots i_{d-2}} = R^T D_{i_1 \dots i_{d-2}} R$ where $D_{i_1 \dots i_{d-2}} = \sum_{j_1 \dots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m, i_m}) S_{j_1 \dots j_{d-2}}$. Therefore $T_{\bar{1}}$ is invertible iff R and $D_{\bar{1}}$ are invertible. Applying Schwartz-Zippel lemma to $\det(R)$ shows that R is singular with probability at most $\frac{n}{|S|}$. We will show that $D_{\bar{1}}$ is singular with probability at most $\frac{n(d-2)}{|S|}$. The lemma then follows from the union bound. Matrix $D_{\bar{1}}$ is not invertible iff $\det(D_{\bar{1}}) = 0$. Since, $D_{\bar{1}} = \sum_{j_1 \dots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m, 1}) S_{j_1 \dots j_{d-2}}$, $\det(D_{\bar{1}}) \in \mathbb{K}[r_{1,1}, \dots, r_{n,1}]$ and $\deg(\det(D_{\bar{1}})) \leq n(d-2)$. Since, \mathcal{V} is not weakly singular, there exists some choice of $\alpha = (\alpha_1, \dots, \alpha_n)$, such that $S = \sum_{i_1, \dots, i_{d-2} \in [n]} (\prod_{m \in [d-2]} \alpha_{i_m}) S_{i_1 \dots i_{d-2}}$ is invertible. Hence, $\det(S) \neq 0$. This gives us that $\det(D_{\bar{1}})(\alpha) \neq 0$, which gives us that $\det(D_{\bar{1}}) \neq 0$. From the Schwartz-Zippel lemma, it follows that $\Pr_{R \in S}[\det(D_{\bar{1}}) = 0] \leq \frac{n(d-2)}{|S|}$. ◀

Recall here from Section 1.4, we define by \mathcal{P}_d , the set of all polynomials of the form $\sum_{i=1}^n \alpha_i x_i^d$ such that $0 \neq \alpha_i \in \mathbb{K}$ for all $i \in [n]$.

► **Lemma 17.** *Given $A \in M_n(\mathbb{K})$, let $\{T_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of $h(x) = P_d(Ax)$ where $P_d \in \mathcal{P}_d$. If $T_{\bar{1}}$ is invertible, define $T_{\bar{1}}' = (T_{\bar{1}})^{-1}$. Then $T_{\bar{1}}' T_{\bar{2}}$ commutes with $T_{\bar{1}}' T_{\bar{3}}$ and $T_{\bar{1}}' T_{\bar{2}}$ is diagonalisable.*

Proof. Let $P_d = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$. By Theorem 4,

$$T_{i_1 \dots i_{d-2}} = A^T (\text{diag}(\alpha_1 (\prod_{m=1}^{d-2} a_{1, i_m}), \dots, \alpha_n (\prod_{m=1}^{d-2} a_{n, i_m}))) A = A^T D_{i_1 \dots i_{d-2}} A.$$

If $T_{\bar{1}}$ is invertible, the same is true of A and $D_{\bar{1}}$. The inverse $(D_{\bar{1}})^{-1}$ is diagonal like $D_{\bar{1}}$, hence $(D_{\bar{1}})^{-1} D_{\bar{2}}$ and $(D_{\bar{1}})^{-1} D_{\bar{3}}$ are both diagonal as well and must therefore commute. Now, $T_{\bar{1}}' T_{\bar{2}} T_{\bar{1}}' T_{\bar{3}} = A^{-1} ((D_{\bar{1}})^{-1} D_{\bar{2}} (D_{\bar{1}})^{-1} D_{\bar{3}}) A = A^{-1} ((D_{\bar{1}})^{-1} D_{\bar{3}} (D_{\bar{1}})^{-1} D_{\bar{2}}) A = T_{\bar{1}}' T_{\bar{3}} T_{\bar{1}}' T_{\bar{2}}$.

Finally, $T_{\bar{1}}' T_{\bar{2}} = A^{-1} ((D_{\bar{1}})^{-1} D_{\bar{2}}) A$ so this matrix is diagonalisable. ◀

We are now in a position to prove the easier half of Theorem 3.

► **Theorem 18.** *If an input $f \in \mathbb{K}[x_1, \dots, x_n]_d$ is equivalent to some polynomial $P_d \in \mathcal{P}_d$ then f will be accepted by Algorithm 1 with high probability over the choice of the random matrix R . More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be accepted with probability $\geq (1 - \frac{n(d-1)}{|S|})$.*

Proof. We start by assuming that $f = P_d(Bx)$ for some $P_d \in \mathcal{P}_d$ where B is an invertible matrix. By Theorem 15, we know that the subspace spanned by the slices of f is not weakly singular. We can therefore apply Lemma 16, the first slice $T_{\bar{1}}$ of $h(x) = f(Rx)$ is invertible with probability at least $1 - \frac{n(d-1)}{|S|}$. Moreover if $T_{\bar{1}}$ is invertible, Lemma 17 shows that, f will always be accepted. (We can apply this lemma to h since $h = P_d(RBx)$). ◀

2.3 Analysis of negative inputs

In this section, we analyse the behaviour of Algorithm 1 on the inputs that are not equivalent to any polynomial in \mathcal{P}_d (which we refer to as the negative inputs). The main goal is to show that the algorithm rejects negative inputs with high probability.

2.3.1 Failure of commutativity

► **Definition 19.** Let $\{S_{i_1, \dots, i_d}\}_{i_1, \dots, i_d \in [n]}$ be a family of matrices. We say that the matrices form a symmetric family of symmetric matrices if each matrix in the family is symmetric and for all permutations $\sigma \in S_d$, $S_{i_1, \dots, i_d} = S_{i_{\sigma(1)} \dots i_{\sigma(d)}}$.

In the next lemma, we show that if a symmetric family of symmetric matrices (this family has size n^d) is not a commuting family, then two linear combinations of these matrices formed by picking just $2n$ elements at random also do not commute with high probability.

► **Lemma 20** (General commutativity lemma). Let $\{S_{i_1, \dots, i_d}\}_{i_1, \dots, i_d \in [n]}$ be a symmetric family of symmetric matrices in $M_n(\mathbb{K})$ that do not form a commuting family. Pick $\alpha = \{\alpha_1, \dots, \alpha_n\}$ and $\alpha' = \{\alpha'_1, \dots, \alpha'_n\}$ uniformly and independently at random from a finite set $S \subset \mathbb{K}$. We define

$$M_\alpha = \sum_{i_1, \dots, i_d \in [n]} \left(\prod_{m \in [d]} \alpha_{i_m} \right) S_{i_1, \dots, i_d} \text{ and } M_{\alpha'} = \sum_{j_1, \dots, j_d \in [n]} \left(\prod_{m \in [d]} \alpha'_{j_m} \right) S_{j_1, \dots, j_d}.$$

Then, $\Pr_{\alpha, \alpha' \in S} [M_\alpha, M_{\alpha'} \text{ don't commute}] \geq \left(1 - \frac{2d}{|S|}\right)$.

Proof. We want to bound the probability of error, i.e $\Pr_{\alpha, \alpha' \in S} [M_\alpha M_{\alpha'} - M_{\alpha'} M_\alpha \neq 0]$. The expression $M_\alpha M_{\alpha'} - M_{\alpha'} M_\alpha$ can be written as $\sum_{i_1, \dots, i_d \in [n]} \left(\prod_{m \in [d]} \alpha_{i_m} \alpha'_{j_m} \right) (S_{i_1 \dots i_d} S_{j_1 \dots j_d} - S_{j_1 \dots j_d} S_{i_1 \dots i_d})$. For a fixed $r, s \in [n]$, we define the polynomial

$$P_{\text{comm}}^{r,s}(\alpha, \alpha') = \sum_{i_1, \dots, i_d, j_1, \dots, j_d \in [n]} \left(\prod_{m \in [d]} \alpha_{i_m} \alpha'_{j_m} \right) m_{i_1 \dots i_d j_1 \dots j_d}^{r,s}$$

where $m_{i_1 \dots i_d j_1 \dots j_d}^{r,s} = (S_{i_1 \dots i_d} S_{j_1 \dots j_d} - S_{j_1 \dots j_d} S_{i_1 \dots i_d})_{r,s}$.

First note that by construction M_α commutes with $M_{\alpha'}$ if and only if for all $r, s \in [n]$ such that $P_{\text{comm}}^{r,s}(\alpha, \alpha') = 0$. Since, $\{S_{i_1, \dots, i_d}\}$ is not a commuting family, there exists $i_1^0, \dots, i_d^0, j_1^0, \dots, j_d^0 \in [n]$, such that $S_{i_1^0 \dots i_d^0} S_{j_1^0 \dots j_d^0} - S_{j_1^0 \dots j_d^0} S_{i_1^0 \dots i_d^0} \neq 0$. Hence, there exists some entry (r_0, s_0) such that $(S_{i_1^0 \dots i_d^0} S_{j_1^0 \dots j_d^0} - S_{j_1^0 \dots j_d^0} S_{i_1^0 \dots i_d^0})_{r_0, s_0} \neq 0$.

Now we claim that $P_{\text{comm}}^{r_0, s_0}(\alpha, \alpha') \neq 0$. It is enough to show that the coefficient of $\alpha_{i_1^0} \dots \alpha_{i_d^0} \alpha'_{j_1^0} \dots \alpha'_{j_d^0}$ in $P_{\text{comm}}^{r_0, s_0}(\alpha, \alpha')$ is non-zero. Let $I_0 = \{(i_{\sigma(1)}^0, \dots, i_{\sigma(d)}^0) | \sigma \in S_d\}$ and let $J_0 = \{(j_{\sigma(1)}^0, \dots, j_{\sigma(d)}^0) | \sigma \in S_d\}$. Then $\text{coeff}_{\alpha_{i_1^0} \dots \alpha_{i_d^0} \alpha'_{j_1^0} \dots \alpha'_{j_d^0}} (P_{\text{comm}}^{r_0, s_0}) = \sum_{\bar{i} \in I_0, \bar{j} \in J_0} m_{\bar{i} \bar{j}}^{r_0, s_0}$. The matrices $S_{i_1 \dots i_d}$ form a symmetric family in the sense of Definition 19. Therefore, for all $\bar{i} \in I_0, \bar{j} \in J_0$, $m_{\bar{i} \bar{j}}^{r_0, s_0}$ are equal. This gives us that $\text{coeff}_{\alpha_{i_1^0} \dots \alpha_{i_d^0} \alpha'_{j_1^0} \dots \alpha'_{j_d^0}} (P_{\text{comm}}^{r_0, s_0}) =$

$|J_0||J_0|(m_{i_1^0 \dots i_d^0 j_1^0 \dots j_d^0}^{r_0, s_0}) \neq 0$. Hence $P_{\text{comm}}^{r_0, s_0} \neq 0$ and $\deg(P_{\text{comm}}^{r_0, s_0}) \leq 2d$ and using Schwartz-Zippel lemma, we get that, $\Pr_{\alpha, \alpha' \in S}[P_{\text{comm}}^{r_0, s_0}(\alpha, \alpha') \neq 0] \geq 1 - \frac{2d}{|S|}$. Putting $r = r_0, s = s_0$, this gives us that $\Pr_{\alpha, \alpha' \in S}[M_\alpha, M_{\alpha'} \text{ don't commute}] \geq \left(1 - \frac{2d}{|S|}\right)$. ◀

The next result relies on the above lemma. Theorem 21 gives us a way to analyze the case when the slices of the input polynomial fail to satisfy the commutativity property (recall that this property is relevant due to Theorem 15).

► **Theorem 21.** *Let $f \in \mathbb{K}[x_1, \dots, x_n]_d$ be a degree d form such that the subspace of matrices \mathcal{V} spanned by its slices is not weakly singular and does not satisfy the commutativity property. Let $h(x) = f(Rx)$ where the entries $(r_{i,j})$ of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$. Let $\{T_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of h . If $T_{\bar{1}}$ is invertible, define $T_{\bar{1}}' = (T_{\bar{1}})^{-1}$. Then $\Pr[T_{\bar{1}}' \text{ is invertible and } T_{\bar{1}}' T_{\bar{2}}, T_{\bar{1}}' T_{\bar{3}} \text{ commute}] \leq \frac{2(d-2)}{|S|}$.*

Proof. Let $\{S_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of f . By Theorem 4, we know that

$$T_{i_1 \dots i_{d-2}} = R^T \left(\sum_{j_1 \dots j_{d-2} \in [n]} \left(\prod_{m \in [d-2]} r_{j_m, i_m} S_{j_1 \dots j_{d-2}} \right) \right) R.$$

Let us define $D_{i_1 \dots i_{d-2}} = \sum_{j_1 \dots j_{d-2} \in [n]} \left(\prod_{m \in [d-2]} r_{j_m, i_m} \right) S_{j_1 \dots j_{d-2}}$. Then we have for all $i \in \{2, \dots, n\}$:

$$T_{\bar{1}}' T_{\bar{i}} = R^{-1} (D_{\bar{1}})^{-1} (R)^{-T} R^T D_{\bar{i}} R = R^{-1} \left(\sum_{j_1 \dots j_{d-2} \in [n]} \left(\prod_{m \in [d-2]} r_{j_m, i} \right) (D_{\bar{1}})^{-1} S_{j_1 \dots j_{d-2}} \right) R. \quad (4)$$

So, if $T_{\bar{1}}$ is invertible, $T_{\bar{1}}' T_{\bar{2}}$ commutes with $T_{\bar{1}}' T_{\bar{3}}$ iff $(D_{\bar{1}})^{-1} D_{\bar{2}}$ commutes with $(D_{\bar{1}})^{-1} D_{\bar{3}}$.

Let E_1 be the event that $T_{\bar{1}}$ is invertible and $T_{\bar{1}}' T_{\bar{2}}$ commutes with $T_{\bar{1}}' T_{\bar{3}}$. Let E_1' be the event that $D_{\bar{1}}$ is invertible and $(D_{\bar{1}})^{-1} D_{\bar{2}}$ commutes with $(D_{\bar{1}})^{-1} D_{\bar{3}}$. Let E_4 be the event that R is invertible. Then we have that $E_1 = E_1' \cap E_4$.

Let E_2 be the event that $D_{\bar{1}}$ is invertible and $\{(D_{\bar{1}})^{-1} S_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ is not a commuting family. Since \mathcal{V} does not satisfy the commutativity property, $(D_{\bar{1}})^{-1} \mathcal{V}$ is not a commuting subspace if $D_{\bar{1}}$ is invertible. Hence, the event that $D_{\bar{1}}$ is invertible is the same as the event E_2 . This also implies that $E_1' \subseteq E_2$. Setting $A_{i_1 \dots i_{d-2}} = (D_{\bar{1}})^{-1} S_{i_1 \dots i_{d-2}}, \alpha_i = r_{i,2}, \alpha_i' = r_{i,3}$ and then using Lemma 20, we can conclude that $\Pr_{R \in S}[E_1' | E_2] \leq \frac{2(d-2)}{|S|}$.

Note here that $D_{\bar{1}}$ depends only on the random variables $r_{i,1}$ for all $i \in [n]$ and therefore is independent of $r_{k,2}$ and $r_{l,3}$ for all $k, l \in [n]$, because we assume that the entries of R are all picked uniformly and independently at random.

Let E_3 be the event that $T_{\bar{1}}$ is invertible. Now we know that $T_{\bar{1}}$ is invertible iff R and $D_{\bar{1}}$ are invertible. Then, we have $E_3 = E_2 \cap E_4$. Hence, the probability of error can be bounded as follows: $\Pr_{R \in S}[E_1] = \Pr_{R \in S}[E_1' \cap E_2 \cap E_4] \leq \Pr_{R \in S}[E_1' | E_2] \leq \frac{2(d-2)}{|S|}$. ◀

2.3.2 Failure of diagonalisability

Theorem 21 gives us a way to analyze the case when the slices of the input polynomial fail to satisfy the commutativity property. With the results in the present section we will be able to analyze the case where the commutativity property is satisfied, but the diagonalisability property fails (recall that these properties are relevant due to Theorem 15).

► **Proposition 22.** *Let $\mathcal{U} \subseteq M_n(\mathbb{K})$ be a commuting subspace of matrices. We define $\mathcal{M} := \{M \mid M \text{ is diagonalisable and } M \in \mathcal{U}\}$. Then \mathcal{M} is a linear subspace of \mathcal{U} . In particular, if there exists $A \in \mathcal{U}$ such that A is not diagonalisable then \mathcal{M} is a proper linear subspace of \mathcal{U} .*

Proof. \mathcal{M} is trivially closed under multiplication by scalars. Let $M, N \in \mathcal{M}$. These two matrices are diagonalisable by definition of \mathcal{M} , and they commute since $\mathcal{M} \subseteq \mathcal{U}$. Hence they are simultaneously diagonalisable. Thus \mathcal{M} is closed under addition as well, which implies that it is a linear subspace of \mathcal{U} . \blacktriangleleft

► **Lemma 23.** *Let $\{A_{i_1 \dots i_d}\}_{i_1, \dots, i_d \in [n]} \in M_n(\mathbb{K})$ be a commuting family of symmetric matrices. Let us assume that this family is symmetric in the sense of Definition 19 and there exists $i_1^0, \dots, i_d^0 \in [n]$ such that $A_{i_1^0 \dots i_d^0}$ is not diagonalisable. Let $S \subset \mathbb{K}$ be a finite set. Then $D = \sum_{i_1, \dots, i_d=1}^n (\prod_{m \in [d]} \alpha_{i_m}) A_{i_1 \dots i_d}$ is diagonalisable with probability at most $\frac{d}{|S|}$ when $\alpha_1, \dots, \alpha_n$ are chosen uniformly and independently at random from S .*

Proof. We define $\mathcal{U} = \text{span}\{A_{i_1 \dots i_d}\}_{i_1, \dots, i_d \in [n]}$. We also define the class of matrices $\mathcal{M} := \left\{ M \mid M \text{ is diagonalisable and } M \in \mathcal{U} \right\}$. So, we want to show that $\Pr_{\alpha \in S} [D \in \mathcal{M}] \leq \frac{d}{|S|}$.

Now using Proposition 22, and the hypothesis that there exists $A_{i_1^0 \dots i_d^0} \in \mathcal{U} \setminus \mathcal{M}$, we get that \mathcal{M} is a proper linear subspace of \mathcal{U} . So \mathcal{M} is an intersection of hyperplanes. Since $A_{i_1^0 \dots i_d^0} \notin \mathcal{M}$, there exists a linear form $l_{\mathcal{M}}(X) = \sum_{i, j \in [n]} a_{ij} X_{ij}$ corresponding to a hyperplane such that $l_{\mathcal{M}}(M) = 0$ for all $M \in \mathcal{M}$ and $l_{\mathcal{M}}(A_{i_1^0 \dots i_d^0}) \neq 0$. We know that if D is diagonalisable, then $l_{\mathcal{M}}(D) \neq 0$. We compute the polynomial $l_{\mathcal{M}}(D)(\alpha) = \sum_{i_1, \dots, i_d \in [n]} (\prod_{m \in [d]} \alpha_{i_m}) m_{i_1 \dots i_d}$ where $m_{i_1 \dots i_d} = (\sum_{k, l \in [n]} a_{kl} (A_{i_1 \dots i_d})_{k, l})$. Now we claim that $l_{\mathcal{M}}(D) \neq 0$. We show this by proving that the coefficient of $\alpha_{i_1^0} \dots \alpha_{i_d^0}$ in $l_{\mathcal{M}}(D)(\alpha)$ is not equal to 0. Let $I_0 = \{(i_{\sigma(1)}^0, \dots, i_{\sigma(d)}^0) \mid \sigma \in S_d\}$. Then $\text{coeff}_{\alpha_{i_1^0} \dots \alpha_{i_d^0}}(D) = \sum_{\bar{i} \in I_0} m_{\bar{i}}$. Since the matrices $A_{i_1 \dots i_d}$ form a symmetric family, the $m_{\bar{i}}$ are equal for all $\bar{i} \in I_0$. Also, since, $l_{\mathcal{M}}(A_{i_1^0 \dots i_d^0}) \neq 0$, we get that $\sum_{k, l \in [n]} a_{k, l} (A_{i_1^0 \dots i_d^0})_{k, l} \neq 0$. This gives us that $m_{i_1^0 \dots i_d^0} \neq 0$. Hence, we get that $\text{coeff}_{\alpha_{i_1^0} \dots \alpha_{i_d^0}}(D) = |I_0| m_{i_1^0 \dots i_d^0} \neq 0$. Thus, $l_{\mathcal{M}}(D) \neq 0$ and $\deg(l_{\mathcal{M}}(D)) \leq d$.

From Schwartz-Zippel lemma, we have $\Pr_{\alpha \in S} [D \in \mathcal{M}] \leq \Pr_{\alpha \in S} [l_{\mathcal{M}}(D)(\alpha) = 0] \leq \frac{d}{|S|}$. \blacktriangleleft

The last result for this section is an analogue of the Theorem 21 for the diagonalisability property.

► **Theorem 24.** *Let $f \in \mathbb{K}[x_1, \dots, x_n]_d$ be a degree- d form such that the subspace \mathcal{V} spanned by its slices is a not weakly-singular subspace, satisfies the commutativity property, but does not satisfy the diagonalisability property. Let $h(x) = f(Rx)$ where the entries $r_{i, j}$ of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$. Let $\{T_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of h . If $T_{\bar{1}}$ is invertible, define $T'_{\bar{1}} = (T_{\bar{1}})^{-1}$. Then $\Pr[T'_{\bar{1}}$ is invertible and $T'_{\bar{1}} T_{\bar{2}}$ is diagonalisable] $\leq \frac{d-2}{|S|}$.*

Proof. As in the proof of Theorem 21, we use the expression for $T'_{\bar{1}} T_{\bar{2}}$ which we obtain from the definition of the slices i.e. $R^{-1} (\sum_{j_1 \dots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m, 2}) (D_{\bar{1}})^{-1} S_{j_1 \dots j_{d-2}}) R$ where $D_{\bar{1}} = \sum_{j_1 \dots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m, 1}) S_{j_1 \dots j_{d-2}}$. So if $T_{\bar{1}}$ is invertible, $T'_{\bar{1}} T_{\bar{2}}$ is diagonalisable iff $M = (\sum_{j_1 \dots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m, 2}) (D_{\bar{1}})^{-1} S_{j_1 \dots j_{d-2}})$ is diagonalisable.

We denote by E_1 the event that $T_{\bar{1}}$ is invertible and $T'_{\bar{1}} T_{\bar{2}}$ is diagonalisable and by E'_1 the event that $D_{\bar{1}}$ is invertible and M is diagonalisable. Let E_4 be the event that R is invertible. Hence, $E_1 = E'_1 \cap E_4$.

Let E_2 be the event that $D_{\bar{1}}$ is invertible and $\{(D_{\bar{1}})^{-1} S_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ is a commuting family and there exists $j_1 \dots j_{d-2} \in [n]$ such that $(D_{\bar{1}})^{-1} S_{j_1 \dots j_{d-2}}$ is not diagonalisable. Since \mathcal{V} satisfies the commutativity property and does not satisfy the diagonalisability property, by Theorem 8, the event that $D_{\bar{1}}$ is invertible is the event same as E_2 . It can also be observed that $E'_1 \subseteq E_2$.

Setting $A_{i_1 \dots i_{d-2}} = (D_{\bar{1}})^{-1} S_{i_1 \dots i_{d-2}}$ and setting $\alpha_i = r_{i,2}$ for all $i \in [n]$ and using Lemma 23, we get that $\Pr_{R \in S}[E'_1 | E_2] \leq \frac{d-2}{|S|}$. Now we know that $T_{\bar{1}}$ is invertible iff R and $D_{\bar{1}}$ is invertible. Let E_3 be the event that $T_{\bar{1}}$ is invertible. Then, we have $E_3 = E_2 \cap E_4$. The probability of error can finally be bounded as follows: $\Pr_{R \in S}[E_1 \cap E_3] = \Pr_{R \in S}[E'_1 \cap E_2 \cap E_4] \leq \Pr_{R \in S}[E'_1 | E_2] \leq \frac{d-2}{|S|}$. \blacktriangleleft

2.3.3 Finishing the analysis for negative inputs

In this section we complete the proof of Theorem 3. The case of positive inputs was treated in Section 2.2. It therefore remains to prove the following result.

► **Theorem 25.** *If an input $f \in \mathbb{K}[x_1, \dots, x_n]_d$ is not equivalent to some polynomial $P_d \in \mathcal{P}_d$, then f is rejected by the algorithm with high probability over the choice of the random matrix R . More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be rejected with probability $\geq (1 - \frac{2(d-2)}{|S|})$.*

Proof. Let $\{S_{i_1, \dots, i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ be the slices of f and $\mathcal{V} = \text{span}\{S_{i_1, \dots, i_{d-2}}\}$. From Theorem 13 and Theorem 9, we know that if $f \not\sim P_d$, then there are three disjoint cases:

1. **Case 1:** \mathcal{V} is a weakly singular subspace of matrices.
2. **Case 2:** \mathcal{V} is not a weakly singular subspace and \mathcal{V} does not satisfy the commutativity property.
3. **Case 3:** \mathcal{V} is not a weakly singular subspace, \mathcal{V} satisfies the commutativity property but does not satisfy the diagonalisability property.

Now we try to upper bound the probability of error in each case.

In case 1, $T_{\bar{1}} = R^T (\sum_{j_1 \dots j_{d-2} \in [n]} r_{j_1,1} \dots r_{j_{d-2},1} S_{j_1 \dots j_{d-2}}) R \in R^T \mathcal{V} R$ is always singular for any choice of $r_{j,1}$. So f is rejected with probability 1 in this case.

In case 2, we can upper bound the probability of error as follows:

$$\begin{aligned} & \Pr_{R \in S}[f \text{ is accepted by the algorithm}] \\ &= \Pr_{R \in S}[T_{\bar{1}} \text{ is invertible, } T_{\bar{1}}' T_{\bar{2}}, T_{\bar{1}}' T_{\bar{3}} \text{ commute, } T_{\bar{1}}' T_{\bar{2}} \text{ is diagonalisable}] \\ &\leq \Pr_{R \in S}[T_{\bar{1}} \text{ is invertible, } T_{\bar{1}}' T_{\bar{2}}, T_{\bar{1}}' T_{\bar{3}} \text{ commute}]. \end{aligned}$$

Using Theorem 21, we get that this occurs with probability at most $\frac{2(d-2)}{|S|}$. In Case 3, we have the following upper bound on the probability of error,

$$\begin{aligned} & \Pr_{R \in S}[f \text{ is accepted by the algorithm}] \\ &= \Pr_{R \in S}[T_{\bar{1}} \text{ is invertible, } T_{\bar{1}}' T_{\bar{2}}, T_{\bar{1}}' T_{\bar{3}} \text{ commute, } T_{\bar{1}}' T_{\bar{2}} \text{ is diagonalisable}] \\ &\leq \Pr_{R \in S}[T_{\bar{1}} \text{ is invertible, } T_{\bar{1}}' T_{\bar{2}} \text{ is diagonalisable}]. \end{aligned}$$

By Theorem 24, this occurs with probability $\leq \frac{d-2}{|S|}$. Therefore in all these three cases, the algorithm rejects f with probability at least $1 - \frac{2(d-2)}{|S|}$. \blacktriangleleft

References

- 1 Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15(80):2773–2832, 2014. URL: <http://jmlr.org/papers/v15/anandkumar14b.html>.
- 2 Alessandra Bernardi, Alessandro Gimigliano, and Monica Ida. Computing symmetric rank for symmetric tensors. *Journal of Symbolic Computation*, 46(1):34–53, 2011.

- 3 Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. *Reconstruction Algorithms for Low-Rank Tensors and Depth-3 Multilinear Circuits*, pages 809–822. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3406325.3451096.
- 4 Aditya Bhaskara, Moses Charikar, Ankur Moitra, and Aravindan Vijayaraghavan. Smoothed analysis of tensor decompositions. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 594–603, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591881.
- 5 L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.
- 6 L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.
- 7 Jérôme Brachat, Pierre Comon, Bernard Mourrain, and Elias Tsigaridas. Symmetric tensor decomposition. *Linear Algebra and its Applications*, 433(11-12):1851–1872, 2010.
- 8 Enrico Carlini. Reducing the number of variables of a polynomial. In *Algebraic geometry and geometric modeling*, Math. Vis., pages 237–247. Springer, Berlin, 2006. doi:10.1007/978-3-540-33275-6_15.
- 9 Guillaume Cheze and André Galligo. Four lectures on polynomial absolute factorization. In *Solving polynomial equations*, pages 339–392. Springer, 2005.
- 10 Guillaume Chèze and Grégoire Lecerf. Lifting and recombination techniques for absolute factorization. *Journal of Complexity*, 23(3):380–420, 2007.
- 11 Shuhong Gao. Factoring multivariate polynomials via partial differential equations. *Mathematics of computation*, 72(242):801–822, 2003.
- 12 Ignacio García-Marco, Pascal Koiran, and Timothée Pecatte. Reconstruction algorithms for sums of affine powers. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 317–324, 2017. doi:10.1145/3087604.3087605.
- 13 Ignacio García-Marco, Pascal Koiran, and Timothée Pecatte. Polynomial equivalence problems for sums of affine powers. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, 2018.
- 14 Ankit Garg, Nikhil Gupta, Neeraj Kayal, and Chandan Saha. Determinant equivalence test over finite fields and over \mathbb{Q} . In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 26, page 42, 2019.
- 15 Richard Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA working papers in phonetics*, 1970.
- 16 Zohar Karnin and Amir Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 274–285, 2009.
- 17 Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, January 2011. URL: <https://www.microsoft.com/en-us/research/publication/efficient-algorithms-for-some-special-cases-of-the-polynomial-equivalence-problem/>.
- 18 Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of full rank algebraic branching programs. *ACM Transactions on Computation Theory (TOCT)*, 11(1):2, 2018.
- 19 Neeraj Kayal and Chandan Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In *Proc. 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–424, 2019.
- 20 Pascal Koiran and Subhayan Saha. Black box absolute reconstruction for sums of powers of linear forms, 2021. doi:10.48550/arXiv.2110.05305.

- 21 Pascal Koiran and Mateusz Skomra. Derandomization and absolute reconstruction for sums of powers of linear forms. *Theoretical Computer Science*, 887:63–84, 2021. doi:10.1016/j.tcs.2021.07.005.
- 22 T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51:455–500, 2009.
- 23 A. Moitra. *Algorithmic Aspects of Machine Learning*. Cambridge University Press, 2018.
- 24 Hani Shaker. Topology and factorization of polynomials. *Mathematica Scandinavica*, pages 51–59, 2009.
- 25 Yaroslav Shitov. How hard is the tensor rank? *arXiv preprint*, 2016. arXiv:1611.01559.
- 26 Amir Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. *SIAM Journal on Computing*, 38(6):2130–2161, 2009.



When You Come at the King You Best Not Miss

Oded Lachish  

Birkbeck College, University of London, UK

Felix Reidl  

Birkbeck College, University of London, UK

Chhaya Trehan  

London School of Economics and Political Science, UK

Abstract

A tournament is an orientation of a complete graph. We say that a vertex x in a tournament \vec{T} controls another vertex y if there exists a directed path of length at most two from x to y . A vertex is called a *king* if it controls every vertex of the tournament. It is well known that every tournament has a king. We follow Shen, Sheng, and Wu [8] in investigating the *query complexity* of finding a king, that is, the number of arcs in \vec{T} one has to know in order to surely identify at least one vertex as a king.

The aforementioned authors showed that one always has to query at least $\Omega(n^{4/3})$ arcs and provided a strategy that queries at most $O(n^{3/2})$. While this upper bound has not yet been improved for the original problem, Biswas et al. [3] proved that with $O(n^{4/3})$ queries one can identify a *semi-king*, meaning a vertex which controls at least half of all vertices.

Our contribution is a novel strategy which improves upon the number of controlled vertices: using $O(n^{4/3} \text{polylog } n)$ queries, we can identify a $(\frac{1}{2} + \frac{2}{17})$ -king. To achieve this goal we use a novel structural result for tournaments.

2012 ACM Subject Classification Mathematics of computing \rightarrow Combinatorial optimization; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Digraphs, tournaments, kings, query complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.25

Funding Chhaya Trehan: Partially supported by EPSRC New Investigator Award EP/V010611/1.

1 Introduction and Related Work

A *tournament* is a directed graph in which there is exactly one directed edge between every pair of vertices. Due to their usefulness in modelling many real world scenarios such as game tournaments, voting strategies and many more, tournaments are a very well studied concept in structural as well as algorithmic graph theory. The early monograph of Moon [7] has been followed by extensive research on the topic. For example, Dey [4] studied the identification of the “best subset of vertices” in a tournament motivated by the high cost of comparing a pair of drugs for a specific disease. Goyal et al. [5] studied the identification of vertices with specific in- or out-degrees.

In this work we investigate the *query complexity* of finding a *king* in a tournament graph, that is, a vertex from which we can reach every other vertex of the tournament via a directed path of length at most two. It is well known that every tournament has such a vertex.

The study of *query complexity* problems in tournaments has the following general shape: Initially, we are only given the vertex set of the tournament while the directions of its arcs are hidden from us. For each pair of vertices u, v we can, at unit cost, learn whether the arc uv or vu is in the tournament. Our goal is to use the fewest possible queries in order to reveal some combinatorial object in the tournament. The motivation for our paper is found in Shen, Sheng, and Wu’s work [8] on the query complexity of identifying a king. They

showed that $\Omega(n^{4/3})$ queries are always necessary and provided an algorithm which reveals a king using $O(n^{3/2})$ queries. Ajtai et al. [1] independently proved the same upper bound in the context of imprecise comparison.

One of the enticing aspects of this setting is its game-theoretic nature: we can alternatively think of it as an adversarial game where one player, the *seeker*, wants to identify a combinatorial structure by querying arcs of the tournament while an adversary, the *obscurer*, tries to delay the seeker for as long as possible by choosing the orientation of queried arcs.

When reading Shen, Sheng, and Wu [8], one may be tempted to conjecture that a better analysis of their obscurer-strategy for finding a king can lead to a better lower bound. However, Biswas et al. [3] showed that against this strategy, the seeker can find a king with $O(n^{4/3})$ queries. They also showed that there exists a seeker strategy with $O(n^{4/3})$ queries for identifying a *semi-king*, that is, a vertex which controls at least half of all vertices. This result is optimal by Lemma 6, Biswas et al. [3]. In fact, one needs to make $\Omega(t^{4/3})$ queries for identifying a vertex which controls at least $t \leq n$ vertices against the obscurer-strategy of Shen, Sheng and Wu. (See Lemma 6 of Biswas et al. [3] and Ajtai et al. [1] for more details.) Therefore, if there exists an obscurer-strategy that proves a stronger than $\Omega(n^{4/3})$ lower bound for the king problem, then this strategy must rely on some factors which distinguish the king problem from the semi-king problem. In our eyes, this means that such a lower bound is much more difficult to find than one might think at first.

Proceeding from the above, it is tempting to try to improve the upper bound by using a variation of the seeker-strategy from Shen, Sheng, and Wu [8] and we can interpret the Biswas et al. [3]’s seeker-strategy for finding a semi-king as such an attempt. These strategies both rely on repeatedly selecting a set of vertices and then querying all the edges between them to find a maximum out-degree (MOD) vertex in this sub-tournament¹. Balasubramanian, Raman and Srinivasaragavan [2] showed that identifying an MOD vertex in a tournament of size k requires $\Omega(k^2)$ queries in the worst case, which may explain the limits of the existing seeker strategies.

Our Result

In this paper, we proceed along the line of research just described. On the one hand, we show that with $\tilde{O}(n^{4/3})$ queries², it is possible to identify a $(\frac{1}{2} + \frac{2}{17})$ -king, which indicates that improving upon the $\Omega(n^{4/3})$ lower bound is probably even harder than indicated by the semi-king results. On the other hand, our technique does not rely on finding MOD vertices of sub-tournaments which circumvents the inherent high cost of this operation.

Technical Overview

Our result is based on the combinatorial structure of tournaments, which may be of independent interest. We believe that this paper provides a novel toolkit which could lead towards resolving the query complexity of finding a king. Specifically, we design a seeker-strategy which consists of two main stages:

- (i) The seeker queries the orientation of a set of edges defined by a so-called *template-graph*. These queries are *non-adaptive* in the sense that the queries do not change as a result of the answers provided by the obscurer.

¹ The relationship between MOD vertices and kings is well-established: Landau [6], while studying the structure of animal societies, showed that every MOD vertex is a king, but non-MOD kings can exist as well.

² The big- \tilde{O} notation hides constants and polylogarithmic factors

- (ii) The seeker analyses the answer to the queries of (i) in order to select queries that lead to the revelation of a $(\frac{1}{2} + \frac{2}{17})$ -king.

The template-graph is an undirected graph over the tournament's vertices that has $\tilde{O}(n^{4/3})$ edges, with the property that every set of vertices of size around $n^{2/3}$ or more has edges to almost all the graph. In Section 3, we use the probabilistic method to prove that such a graph exists. Given the template-graph, the seeker's queries in the first stage are simply given by its edges, i. e. if there exists an edge uv in the template-graph, then the seeker asks the obscurer about the orientation of the edge uv in the tournament. The sparsity of the template-graph ensures that the seeker does not make too many queries and the connectivity of every sufficiently large set ensures that we do not miss any relevant information.

The second stage of the seeker-strategy is built on showing that when the obscurer chooses how the edges of the template graph are oriented they have a trade-off. The trade-off is either to reveal an *ultra-set* or not. We show that if the obscurer reveals an *ultra-set*, then the seeker can reveal a $(\frac{1}{2} + \frac{2}{17})$ -king with $\tilde{O}(n^{4/3})$ extra queries. If the obscurer does not do this, then the seeker can use this to find a partition of the vertex set of the tournament into sets of size $O(n^{2/3})$ each (which we refer to as *tiles*), so that the edges of the template-graph that are incident to the tiles satisfy a certain property. We obtain this combinatorial object by showing that if such a partition does not exist, then a simple set of queries already reveals a $(\frac{1}{2} + \frac{2}{17})$ -king.

The tiles are analysed by the construction of what we refer to as the *free matrix* which contains a row for every tile and a column for every vertex of the tournament. An entry of the matrix indexed by a given tile-vertex pair is 1 if every edge between the vertex and a vertex in the tile is directed towards the tile, otherwise the entry is 0. We then use this free matrix to guide the seeker-strategy.

Given that the first part of the seeker-strategy is non-adaptive and against any adversary, this approach also reveals a combinatorial property of tournaments: for any fixed tournament and template graph with the same set of vertices, knowing only the direction of the arcs of the tournament that correspond to the arcs of the template graph is sufficient for finding a set of vertices S of size $O(n^{2/3})$ such that querying all edges inside of S necessarily reveals a $(\frac{1}{2} + \frac{2}{17})$ -king. We note that fraction $\frac{2}{17}$ is the result of balancing the various trade-offs in the seeker strategy.

The rest of the paper is organised as follows. In Section 2, we provide necessary definitions and prove some basic lemmas about tournaments that are used in the rest of the paper. Section 3 is dedicated to the formal definition and the proof of existence of template graphs. In Section 4, we describe our seeker-strategy and prove that it leads to the discovery of a $(\frac{1}{2} + \frac{2}{17})$ -king. In Section 5, we give concluding remarks and open problems.

2 Preliminaries

For simplicity, we assume that the vertices of any n -vertex graph are the numbers $[n] := \{1, \dots, n\}$.

An orientation of a graph G is a directed graph obtained from G by replacing every one of its edges by a directed arc.

$d(v, X), N(v)$ Given an undirected graph G , a vertex $v \in V(G)$, and a vertex set $X \subseteq V(G)$ we define the *relative degree* $d(v, X) := |N(v) \cap X|$, where $N(v)$ is the neighbourhood of v in G .

$d^+(X), N^+(v)$ For a vertex v in a directed graph \vec{G} , a vertex u in \vec{G} is an out-neighbour of v , if the edge between u and v is oriented from v to u . For a directed graph \vec{G} , we denote the out-neighbourhood of a vertex $v \in \vec{G}$ by $N^+(v)$ and its out-degree by $d^+(v)$. For a vertex

25:4 When You Come at the King You Best Not Miss

set $X \subseteq V(\vec{G})$, we let $d^+(X)$ be the number of arcs from a vertex in X to a vertex not in X . $d^+(v, X)$ Given additionally a vertex subset $X \subseteq V(G)$, we define the *relative out-degree* $d^+(v, X) := |N^+(v) \cap X|$.

$N^{++}[v]$ The (closed) *second-out-neighbourhood* $N^{++}[v]$ of v is the set of vertices $u \in V(\vec{G})$ for which there exists a directed path from v to u of length at most two.

control, direct – For simplicity we will adopt the following vocabulary for digraphs. We say that a vertex x *controls* a vertex y if $y \in N^{++}[x]$. We say that x *directly controls* a vertex y if $y \in N^+(x) \cup \{x\}$. We extend both of these terms to vertex sets U , for example, we will often write statements like “ x controls at least half of the vertices in U ”.

$\vec{T}, \vec{T}[S]$ A tournament is a digraph \vec{T} obtained from a complete graph by replacing each edge with a directed arc. As done usually, we denote the subgraph induced by a vertex set $S \subseteq V(\vec{T})$, with $\vec{T}[S]$. Note that an induced subgraph of a tournament is necessarily also a tournament. We will need the following basics facts about tournaments in the following.

► **Lemma 1.** *Let \vec{T} be a tournament with m vertices and $\alpha \in [0, 1]$ such that αm is even. Then \vec{T} has at least $(1 - \alpha)m$ vertices of out-degree at least $\alpha m/2$.*

Proof. Let S initially be the vertices of \vec{T} and proceed according to the following process: find a vertex of out-degree at least $\alpha m/2$ and remove it from S ; and repeat until no such vertex exists in S . From here on we focus on set S after the vertex removal process ended.

Let $r = |S|$ be the size of the final set and consider the sub-tournament $\vec{T}[S]$. We know that by averaging considerations, every tournament of size r has at least one vertex of out-degree at least $r/2 - 1/2$. We also know that S does not contain any vertex of out-degree at least $\alpha m/2$. Hence, we conclude that $r \leq \alpha m$.

Consequently, our process discovered $m - r \geq (1 - \alpha)m$ vertices of out-degree at least $\alpha m/2$ in \vec{T} . ◀

► **Lemma 2.** *Let \vec{G} be an orientation of a complete bipartite graph (V_0, V_1, E) , where $|V_0| = |V_1| = m$, and m is divisible by 4. Then, there exists $i \in \{0, 1\}$, such that V_i has at least $m/2 + 1$ vertices v , where $d^+(v, V_{1-i}) \geq m/4$.*

Proof. Let S initially contain all the vertices in $V_0 \cup V_1$ and proceed according to the following process: We find a vertex of out-degree at least $m/4$ and remove it from S . Repeat until no such vertex exists and we are left with $S' \subseteq S$.

Every orientation of the complete bipartite graph $K_{t,t}$ must contain, by a simple averaging argument, a vertex of out-degree at least $t/2$. Therefore the induced subgraph $\vec{G}[S']$ must have at least one partite set of size strictly less than $m/2$ or we could continue the process. Consequently, our process discovered at least $m/2 + 1$ vertices of out-degree at least $m/4$ in that partite set. ◀

► **Lemma 3.** *Let \vec{T} be a tournament on $2m$ vertices, where m is divisible by 4. Let further sets S_0, S_1 be a partition of the vertices of \vec{T} into sets of equal size. Then there exists a vertex v such that both $d^+(v, S_0) \geq m/4$ and $d^+(v, S_1) \geq m/4$.*

Proof. By Lemma 1, for both $i \in \{0, 1\}$, there exist $m/2$ vertices v in S_i such that $|N^+(v) \cap S_i| \geq m/4$. By Lemma 2 for *one* of $i \in \{0, 1\}$, there exist $m/2 + 1$ vertices v such that $|N^+(v) \cap S_{1-i}| \geq m/4$. Then by the pigeonhole principle, there exists $i \in \{0, 1\}$ and a vertex $v \in S_i$, such that $|N^+(v) \cap S_i| \geq m/4$, for every $i \in \{0, 1\}$, as claimed. ◀

3 Constructing the template-graph

► **Definition 4** (κ -template-graph). Let $\kappa \in (0, 1)$ and G be an undirected graph over the vertex set $[n]$. The graph G is a κ -template-graph, if for every pair of disjoint sets $H_1, H_2 \subseteq [n]$ both of size at least $\kappa n^{2/3}$, there exists at least one edge between them, that is, $|E(H_1, H_2)| \geq 1$.

For the remainder of this section, we fix $\kappa \in (0, 1)$ and set $p = \frac{2 \log n + 2}{\kappa n^{2/3}}$. We next show that with strictly positive probability the Erdős–Renyi random graph $G(n, p)$ is a κ -template-graph, with $O(n^{4/3} \log n)$ edges, where the O notation hides a dependence on κ . By the probabilistic method, this implies that there actually exists such a graph.

All probabilities in the following are with respect to the probability space of this random graph.

► **Lemma 5.** Let $\kappa \in (0, 1)$. With probability at least $3/4$, the graph $G(n, p)$, where p is defined as above, is a κ -template-graph.

Proof. Note that if we prove the statement of the lemma for sets of size *exactly* (up to rounding errors) $\kappa n^{2/3}$, then the claim follows for all larger sets as well. To prove this, we next show, with the help of the union bound, that the probability that $G(n, p)$ has two disjoint subsets of vertices, each of size $\kappa n^{2/3}$, with no edge between them is strictly less than $1/4$.

Let H_1 and H_2 be any pair of disjoint subsets of $[n]$ of size $\kappa n^{2/3}$, then the total number of vertex pairs between them is $(\kappa n^{2/3})^2$. The probability that none of these pairs is an edge in the template-graph G is accordingly $(1 - p)^{(\kappa n^{2/3})^2}$.

We apply the exponential bound $(1 - p)^k \leq e^{-pk}$ for a k -round Bernoulli trial and obtain

$$\begin{aligned} (1 - p)^{(\kappa n^{2/3})^2} &\leq e^{-p(\kappa n^{2/3})^2} = e^{-(2 \log n + 2)\kappa n^{2/3}} \\ &= n^{-2\kappa n^{2/3}} e^{-2\kappa n^{2/3}} \leq \frac{1}{4} n^{-2\kappa n^{2/3}}, \end{aligned}$$

where the last inequality holds when n is large enough so that $\kappa n^{2/3} \geq 1$ and hence $e^{-2\kappa n^{2/3}} < \frac{1}{4}$. Since the total number of pairs of sets H_1, H_2 of size $\kappa n^{2/3}$ is bounded above by $n^{2\kappa n^{2/3}}$, the claim now follows from the union bound. ◀

► **Theorem 6.** Let $\kappa \in (0, 1)$, there exists a κ -template-graph G with at most $O(n^{4/3} \log n / \kappa)$ edges.

Proof. The expected number of edges of $G(n, p)$ for our choice of $p = (2 \log n + 2) / (\kappa n^{2/3})$ is less than $m := (2 \log n + 2)n^{4/3} / \kappa$. Since every edge of the graph is selected independently, by the Chernoff bound the probability that the number of edges in $G(n, p)$ exceeds $2m$ is at most

$$\begin{aligned} e^{-p \binom{n}{2} / 3} &\leq e^{-\frac{(2 \log n + 2) n(n-1)}{\kappa n^{2/3} \cdot 6}} \\ &\leq e^{-(2 \log n + 2)n/6} \leq \frac{1}{4} \end{aligned}$$

where the last inequality holds for $n \geq 4$.

Together with Lemma 5 this implies that with probability at least $1/2$, $G(n, p)$ is a κ -template-graph G with at most $O(n^{4/3} \log n / \kappa)$ edges. The claim follows by the probabilistic method. ◀

4 The seeker strategy

Having proved the existence of a κ -template-graph, we next examine the properties of an arbitrary orientation of such a graph. Given a κ -template-graph G_κ on a vertex set $[n]$, we use \vec{G}_κ (*henceforth*) to refer to a directed graph obtained by replacing every edge of G_κ by a directed arc. Note that we assume nothing about \vec{G}_κ and analyze as if its arcs were arbitrarily oriented by an adversary.

► **Definition 7** (η -weak, η -strong, η -ultra). *η -weak, η -strong, η -ultra* For an oriented template-graph \vec{G}_κ and any $\eta > 0$, a set $H \subseteq [n]$ is η -weak if $d^+(H) < (1/2 + \eta)n$, is η -strong if $d^+(H) \geq (1/2 + \eta)n$. We call a set η -ultra if every subset $H' \subset H$, of size at least $|H|/2$ is η -strong.

To understand why η -ultra sets are important, it is useful to think of the seeker as trying to force the obscurer to reveal enough information (in the form of query answers) so that the seeker can achieve their goal. This is done by first querying the orientation of all the edges of a template graph and nothing else. The observation below implies that if the orientation of the edges of the template graphs reveals an η -ultra set, of size $\tilde{O}(n^{2/3})$, then the seeker can achieve its goal with an additional $\tilde{O}(n^{4/3})$ queries. Thus, the obscurer cannot reveal such an ultra-set. However, as we show further on, by doing this the obscurer reveals enough information for the seeker to achieve their goal.

► **Observation 8.** *Let $H \subseteq V(\vec{G}_\kappa)$ be an η -ultra set. Then we can find a $(1/2 + \eta)$ -king using $\leq |H|^2$ additional queries.*

Proof. Query all $\leq |H|^2$ edges inside H . Let $v \in H$ be a vertex such that $d^+(v, H) \geq |H|/2$. Since H is η -ultra, the set $H' := N^+(v) \cap H$ is η -strong, meaning $d^+(H') \geq (1/2 + \eta)n$. Therefore $|N^{++}[v]| \geq (1/2 + \eta)n$ and v is a $(1/2 + \eta)$ -king. ◀

► **Definition 9** (Free set). *Free set, $F(W)$* Let $W \subseteq V(\vec{G}_\kappa)$ be an η -weak set. Then the free set of W is the vertex set $F(W) := V(\vec{G}_\kappa) \setminus (N^+(W) \cup W)$, that is, all vertices that lie neither in W nor in $N^+(W)$.

► **Observation 10.** *Let W be an η -weak set. Then $|F(W)| > (\frac{1}{2} - \eta)n - |W|$.*

By the properties of template-graphs, namely that each pair of large enough sets must have an edge between them, and by the definition of free sets it follows that all the arcs of \vec{G}_κ between a sufficiently large set W and its free set $F(W)$ must point towards W . Let us formalize this intuition:

► **Definition 11** (α -covers). *For $\alpha \in [0, 1]$ we say that a set S α -covers a set W if $|N^+(S) \cap W| \geq \alpha|W|$.*

► **Lemma 12.** *In the template graph, for every set $W \subset [n]$ of size $n^{2/3}$ and every subset $S \subseteq F(W)$ of size at least $\kappa n^{2/3}$, it holds that S $(1 - \kappa)$ -covers W .*

Proof. Consider a set $S \subseteq F(W)$ of size $\kappa n^{2/3}$. Then, by Lemma 5, there is at least one edge $s_1 w_1$ between some $s_1 \in S$ and $w_1 \in W$. Remove w_1 from W and apply the argument to the remainder. In this way, we construct a sequence w_1, \dots, w_t such that each w_i has at least one neighbour in S .

The application of Lemma 5 is possible until the remainder of W has size less than $\kappa n^{2/3}$, hence the process works for at least $t = n^{2/3} - \kappa n^{2/3} = (1 - \kappa)n^{2/3}$ steps. Now simply note that each edge $s w_i$ for $s \in S$ must be oriented from s to w_i since S is a subset of $F(W)$. It follows that $|N^+(S) \cap W| \geq (1 - \kappa)|W|$, as claimed. ◀

In the previous lemma lies the inherent usefulness of free sets. If, for some set W of size $n^{2/3}$, we find a vertex v that has at least $\kappa n^{2/3}$ out-neighbours in the free set $F(W)$, then v controls almost all of W . As observed above, η -weak sets have necessarily large free sets which makes them “easy targets” for our strategy.

We now show that in case no η -ultra set exists (in which case we already win as per Observation 8), we can instead partition most of the vertices of $V(\vec{G}_\kappa)$ into weak sets.

► **Definition 13.** An η -weak tiling of \vec{G}_κ is a vertex partition W_1, \dots, W_m, R where $|W_i| = n^{2/3}$, $|R| < 2n^{2/3}$ and every set W_i is η -weak. We call the sets W_i the tiles and R the remainder.

By definition, the number of tiles m in an η -weak tiling is at least $n^{1/3} - 2$.

► **Lemma 14.** Fix $\eta > 0$. For large enough n , \vec{G}_κ either contains an η -ultra set of size $2n^{2/3}$ or an η -weak tiling.

Proof. We construct the tiling iteratively. Assume we have constructed W_1, \dots, W_j so far. Let $R := V(\vec{G}_\kappa) \setminus \bigcup_{i \leq j} W_i$ be all the vertices of \vec{G}_κ which are not yet part of the tiling. If $|R| < 2n^{2/3}$ we are done, so assume otherwise. Let $H \subseteq R$ be an arbitrary vertex set of size $2n^{2/3}$. If H is η -ultra, then by Observation 8, we are done. Otherwise there exists an η -weak set $W_{j+1} \subseteq H$, $|W_{j+1}| = |H|/2$. Add this set to the tiling and repeat the construction. At the end of this procedure, we will either find an η -ultra set or an η -weak tiling. ◀

Our goal is now to find a vertex whose out-neighbourhood has large intersections with many free sets. To organise this search, we define the following auxiliary structure:

► **Definition 15 (Free matrix).** Let W_1, \dots, W_m, R be an η -weak tiling of \vec{G}_κ and let $\mathcal{W} = \{W_1, \dots, W_m\}$. The free matrix M of the tiling \mathcal{W}, R is a binary matrix with m rows indexed by \mathcal{W} and n columns indexed by $[n]$. The entry at position $(W_i, v) \in \mathcal{W} \times V$ is 1 if $v \in F(W_i)$ and 0 otherwise.

weight, $\sum M$ We will use the following notation in the rest of this section. Given a free matrix M of an η -weak tiling \mathcal{W}, R let $M[\mathcal{W}', U]$ denote a sub-matrix of M induced by a subset $\mathcal{W}' \subseteq \mathcal{W}$ of the tile set and a subset $U \subseteq [n]$ of the vertex set. For example, a column of M corresponding to a vertex $v \in [n]$ can be written as $M[\mathcal{W}, \{v\}]$ in this notation. Analogously a row of M corresponding to a tile $W_i \in \mathcal{W}$ can be written as $M[\{W_i\}, [n]]$. Given a sub-matrix M' of the free matrix M , we call the number of 1's in M' the *weight* of M' and denote it by $\sum M'$.

The following is a direct consequence of the construction of the free matrix and Observation 10.

► **Observation 16.** Every row of the free matrix M has a weight of at least $(\frac{1}{2} - \eta - n^{-1/3})n$.

► **Definition 17 (Good Sub-Matrix).** A sub-matrix $M[\mathcal{W}, U]$, for some $U \subset [n]$, is η -good if, each one of its rows has weight at least $(\frac{1}{2} - \eta - 2n^{-1/3} \log^{1/2} n)|U|$.

We next show that a good sub-matrix with $2n^{2/3}$ columns exists, by using the probabilistic method. Specifically, we show that if we randomly pick $2n^{2/3}$ columns from the matrix $M[\mathcal{W}, [n]]$ then with strictly positive probability the matrix that includes exactly these columns is good.

► **Lemma 18.** Let $\eta \in (0, \frac{1}{2})$. For large enough n the free matrix M has an η -good sub-matrix with $2n^{2/3}$ columns.

Proof. Select $K \subset [n]$ of size $2n^{2/3}$ uniformly at random. Let $M' = M[\mathcal{W}, K]$.

We set $p = 1/2 - \eta - n^{-1/3}$ and $t = n^{-1/3} \log^{1/2} n$. By Observation 16, every row of M has weight at least pn . By the Hypergeometric tail bound the probability that a specific row of M' has weight less than $(p-t)n$ is at most $e^{-2t^2 2n^{2/3}} \leq 1/n$, where the inequality follows from our choice of t . Then by the union bound the probability that *every* row of M' has weight at least $(p-t)n$ is strictly positive.

Note that by our choice of p and t , we get that $(p-t)n$, for large enough n , is at least as large as $(1/2 - \eta - 2n^{-1/3} \log^{1/2} n)n$, therefore a good sub-matrix M' exists with strictly positive probability. By the probabilistic methods the claim therefore holds. \blacktriangleleft

Next we show that the only way that the adversary does not provide us with a $(1/2 + \delta)$ -king once we have identified a δ -good sub-matrix is if the distribution of 1's in that matrix is very restricted. We will use this additional structure to find a $(1/2 + \delta)$ -king in the sequel. For simplicity, we first query all the edges between the vertices associated with the columns of the δ -good sub-matrix, but note that this is not strictly necessary: we can instead inspect all potential partitions (with properties as stated in the lemma) and only if no such partition exists query said edges which then surely identifies a $(1/2 + \delta)$ -king. With this change the lemma is consistent with the structural claim from the introduction.

► **Lemma 19.** *Let $M' = M[\mathcal{W}, V]$ be a δ -good sub-matrix of M with $2n^{2/3}$ columns. Let further $\kappa + \delta \leq 1/2$. If we query each one of the $O(n^{4/3})$ edges in V , then either we find a $(\frac{1}{2} + \delta)$ -king, or we find partitions $V_1 \uplus V_2 = V$ and $\mathcal{W}_1 \uplus \mathcal{W}_2 = \mathcal{W}$ with the following properties:*

- $|V_1| = |V_2| = n^{2/3}$
- $|\mathcal{W}_1| \geq (\frac{1}{2} - \delta - \kappa)n^{1/3} - 2$ and $|\mathcal{W}_2| < (\frac{1}{2} + \delta + \kappa)n^{1/3}$
- Every row in $M'[\mathcal{W}_1, V_1]$ has weight at most $\kappa n^{2/3}$
- Every row in $M'[\mathcal{W}_2, V_1]$ has weight at least $\kappa n^{2/3}$

Proof. We query all the edges in $V \times V$ and select a vertex $y \in V$ such that $d^+(y, V) \geq n^{2/3}$. Let V_1 be an arbitrary subset of $N^+(y) \cap V$ of size $n^{2/3}$ and let $V_2 = V \setminus V_1$. Partition the rows of M' into $\mathcal{W}_1 \cup \mathcal{W}_2$ so that \mathcal{W}_1 contains all rows with weight less than $\kappa n^{2/3}$ in the sub-matrix $M'[\mathcal{W}, V_1]$.

We claim that if $|\mathcal{W}_1| < (\frac{1}{2} - \delta - \kappa)n^{1/3} - 2$ then y is a $(\frac{1}{2} + \delta)$ -king. By construction, every row in \mathcal{W}_2 has weight at least $\kappa n^{2/3}$ in $M'[\mathcal{W}, V_1]$ and if the condition of the claim holds then $|\mathcal{W}_2| \geq (\frac{1}{2} + \delta + \kappa)n^{1/3}$. By Lemma 12, the set V_1 must $(1 - \kappa)$ -cover every tile in \mathcal{W}_2 . It follows that

$$\begin{aligned} |N^{++}[y]| &\geq (1 - \kappa) |\bigcup \mathcal{W}_2| \geq (1 - \kappa) \left(\left(\frac{1}{2} + \delta + \kappa \right) n^{1/3} \right) n^{2/3} \\ &= (1 - \kappa) \left(\frac{1}{2} + \delta + \kappa \right) n = \left(\frac{1}{2} + \delta + \frac{\kappa}{2} - \kappa\delta - \kappa^2 \right) n \\ &= \left(\frac{1}{2} + \delta + \kappa \left(\frac{1}{2} - \delta - \kappa \right) \right) n \\ &\geq \left(\frac{1}{2} + \delta \right) n \end{aligned}$$

where the last inequality holds since $\delta + \kappa \leq 1/2$. \blacktriangleleft

Lemma 19 implies the following about good sub-matrices.

► **Lemma 20.** *Let $M' = M[\mathcal{W}, V]$ be a δ -good sub-matrix with $|V| = 2n^{2/3}$ and $\mathcal{W}_1 \uplus \mathcal{W}_2, V_1 \uplus V_2$ be partitions as in Lemma 19. Then every row in $M'[\mathcal{W}_1, V_2]$ has weight at least $(1 - 2\delta - 2\kappa)n^{2/3}$.*

Proof. Since M' is a δ -good sub-matrix, by Definition 17, every row in $M'[\mathcal{W}_1, V]$ has weight at least $(1/2 - \delta - 2n^{-1/3} \log^{1/2} n)2n^{2/3}$. By Lemma 19, every row in $M'[\mathcal{W}_1, V_1]$ has weight at most $\kappa n^{2/3}$. Therefore, the weight of every row in $M'[\mathcal{W}_1, V_2]$ is

$$\begin{aligned} &\geq (1/2 - \delta - 2n^{-1/3} \log^{1/2} n)2n^{2/3} - \kappa n^{2/3} \\ &= (1 - 2\delta - 4 \frac{\log^{1/2} n}{n^{1/3}} - \kappa)n^{2/3} \\ &\geq (1 - 2\delta - 2\kappa)n^{2/3} \end{aligned}$$

where we assume that n is large enough so that $4 \frac{\log^{1/2} n}{n^{1/3}} \leq \kappa$. \blacktriangleleft

Our final technical lemma lets us, for a given set of rows of M , identify a set of columns with high enough weight when restricted to those rows.

► **Lemma 21.** *Let $U \subset [n]$ be of size $2n^{2/3}$ and $\mathcal{W}' \subset \mathcal{W}$. Then there exists a set $V' \subset [n] \setminus U$, of size $n^{2/3}$ such that every column in $M[\mathcal{W}', V']$ has weight at least $(1/2 - \delta - 3n^{-1/3})|\mathcal{W}'|$.*

Proof. Let $\bar{U} := [n] \setminus U$ and let V' be an arbitrary subset of $n^{2/3}$ columns in $M[\mathcal{W}', \bar{U}]$ with the largest column-weight. Let t be the smallest weight among these columns when restricted to $M[\mathcal{W}', V']$. We bound the weight of $M[\mathcal{W}', \bar{U}]$ first from below and then from above, then we use these bounds to show that $t > (\frac{1}{2} - \delta - 3n^{-1/3}) \cdot |\mathcal{W}'|$, which implies that V' is the claimed set.

For the lower bound on the weight of $M[\mathcal{W}', \bar{U}]$, we use the simple fact that

$$\sum M[\mathcal{W}', \bar{U}] = \sum M[\mathcal{W}', [n]] - \sum M[\mathcal{W}', U]. \quad (1)$$

By Observation 16, every row of the matrix M has weight least $(\frac{1}{2} - \delta - n^{-1/3}) \cdot n$. It follows that $\sum M[\mathcal{W}', [n]]$ is at least $(\frac{1}{2} - \delta - n^{-1/3}) \cdot n \cdot |\mathcal{W}'|$. For the second term, we have the trivial bound $\sum M[\mathcal{W}', U] \leq |U| \cdot |\mathcal{W}'| = 2n^{2/3} \cdot |\mathcal{W}'|$. Plugging these values into (1) we obtain

$$\begin{aligned} \sum M[\mathcal{W}', \bar{U}] &\geq (\frac{1}{2} - \delta - n^{-1/3}) \cdot n \cdot |\mathcal{W}'| - 2n^{2/3} \cdot |\mathcal{W}'| \\ &= (\frac{1}{2} - \delta - 3n^{-1/3}) \cdot n \cdot |\mathcal{W}'|. \end{aligned} \quad (2)$$

For the upper bound on the total weight of $M[\mathcal{W}', \bar{U}]$ we use that

$$\sum M[\mathcal{W}', \bar{U}] = \sum M[\mathcal{W}', V'] + \sum M[\mathcal{W}', \bar{U} \setminus V']. \quad (3)$$

We use the trivial bound $\sum M[\mathcal{W}', V'] \leq |V'| \cdot |\mathcal{W}'| = n^{2/3} \cdot |\mathcal{W}'|$ for the first term. By definition of the value t , we have that every column in $M[\mathcal{W}', \bar{U} \setminus V']$ has weight at most t . Accordingly, $\sum M[\mathcal{W}', \bar{U} \setminus V'] \leq t \cdot |\bar{U} \setminus V'| = t \cdot (n - 3n^{2/3})$. Plugging in these values into (3) we obtain

$$\sum M[\mathcal{W}', \bar{U}] \leq n^{2/3} \cdot |\mathcal{W}'| + t \cdot (n - 3n^{2/3}). \quad (4)$$

Finally, (2) and (4) taken together give us that

$$t \cdot (n - 3n^{2/3}) + n^{2/3} \cdot |\mathcal{W}'| \geq (\frac{1}{2} - \delta - 2n^{-1/3}) \cdot n \cdot |\mathcal{W}'|.$$

Consequently, $t > (\frac{1}{2} - \delta - 3n^{-1/3}) \cdot |\mathcal{W}'|$ and we conclude that V' has the claimed property. \blacktriangleleft

25:10 When You Come at the King You Best Not Miss

We are finally ready to prove our seeker-strategy for finding a $1/2 + \delta$ -king using $\tilde{O}(n^{4/3})$ queries. For readability, we will state our main result in terms of concrete and simple values for κ and δ , however, note that smaller values of κ allow δ to be slightly larger than the stated bound of $\frac{2}{17}$.

► **Theorem 22.** Fix $\delta = \frac{2}{17}$, let $\kappa = \frac{1}{4000}$. For large enough n , there exists a seeker strategy for finding a $(1/2 + \delta)$ -king using $\tilde{O}(n^{4/3})$ edge queries.

Proof. We construct the template-graph G_κ and query all $\tilde{O}(n^{4/3})$ of its edges to obtain \vec{G}_κ .

By Lemma 14, we either obtain a δ -ultra set of size $2 \cdot n^{2/3}$ or a δ -weak tiling of \vec{G}_κ . If we find the former, by Observation 8 we can find a $(\frac{1}{2} + \delta)$ -king using $O(n^{4/3})$ additional queries. Therefore assume that we obtained a δ -weak tiling \mathcal{W}, R of \vec{G}_κ .

Let M be the free matrix of \mathcal{W}, R . By Lemma 18, M has a δ -good sub-matrix $M[\mathcal{W}, V]$ with $|V| = 2n^{2/3}$. We query all $O(n^{4/3})$ edges in $V \times V$ and by Lemma 19 either identify a $(\frac{1}{2} + \delta)$ -king, or obtain partitions $V_1 \uplus V_2 = V$, $\mathcal{W}_1 \uplus \mathcal{W}_2$ with properties as listed in Lemma 19. Importantly, by Lemma 20, every row in the sub-matrix $M[\mathcal{W}_1, V_2]$ has weight at least $(1 - 2\delta - 2\kappa)n^{2/3}$.

We now apply Lemma 21 with $\mathcal{W}' = \mathcal{W}_2$ and find a set of columns $V_3 \subseteq [n] \setminus V$ of size $n^{2/3}$ such that every column in $M[\mathcal{W}_2, V_3]$ has weight at least $(\frac{1}{2} - \delta - 3n^{-1/3})|\mathcal{W}_2|$. We now query all edges in $V_3 \times V_3$ and $V_2 \times V_3$, since $|V_2| = |V_3| = n^{2/3}$ this amounts to $O(n^{4/3})$ additional queries.

Since $\vec{G}[V_2 \cup V_3]$ is completely revealed, it is a tournament of size $2n^{2/3}$ and we apply Lemma 3 using the bipartition (V_2, V_3) to find a vertex $v \in V_2 \cup V_3$ such that $d^+(v, V_2)$ and $d^+(v, V_3)$ are both at least $n^{2/3}/4$. We claim that v is a $(\frac{1}{2} + \delta)$ -king. Let in the following $V'_2 = N^+(v) \cap V_2$ and $V'_3 = N^+(v) \cap V_3$. We first prove the following two claims about these two sets:

▷ **Claim 23.** Every row in $M[\mathcal{W}_1, V'_2]$ has weight at least $\kappa n^{2/3}$.

Proof of the claim. According to Lemma 20, every row in $M[\mathcal{W}_1, V_2]$ has weight at least $(1 - 2\delta - 2\kappa)n^{2/3}$. Since $|V'_2| = |V_2|/4 = n^{2/3}/4$, we have that each row in $M[\mathcal{W}_1, V'_2]$ has weight at least

$$(1 - 2\delta - 2\kappa)n^{2/3} - \frac{3}{4}n^{2/3}$$

which is larger than $\kappa n^{2/3}$ for $\delta \leq \frac{1}{8} - \frac{3\kappa}{2}$ which holds true for our choices of δ and κ . ◁

▷ **Claim 24.** At least $(\frac{1}{2} - \delta - 4\kappa - 3n^{-1/3})\frac{|\mathcal{W}_2|}{1-4\kappa}$ rows in $M[\mathcal{W}_2, V'_3]$ have weight at least $\kappa n^{2/3}$.

Proof of the claim. Recall that by choice of V_3 , every column in $M[\mathcal{W}_2, V_3]$ and therefore also $M[\mathcal{W}_2, V'_3]$ has weight at least $(\frac{1}{2} - \delta - 3n^{-1/3})|\mathcal{W}_2|$. Accordingly,

$$\begin{aligned} \sum M[\mathcal{W}_2, V'_3] &\geq (\frac{1}{2} - \delta - 3n^{-1/3})|\mathcal{W}_2| \cdot |V'_3| \\ &\geq (\frac{1}{2} - \delta - 3n^{-1/3})|\mathcal{W}_2| \cdot \frac{1}{4}n^{2/3}. \end{aligned} \tag{5}$$

Let t denote the number of rows in $M[\mathcal{W}_2, V'_3]$ with weight at least $\kappa n^{2/3}$. Our goal is to find a lower bound for t . Since t is minimized if every row that has weight at least $\kappa n^{2/3}$ has in fact the maximum possible weight $|V'_3| = n^{2/3}/4$, we can lower-bound t using

$$\frac{t}{4}n^{2/3} + (|\mathcal{W}_2| - t)\kappa n^{2/3} \geq \sum M[\mathcal{W}_2, V'_3].$$

Combining this inequality with (5), we obtain

$$\begin{aligned} t \frac{n^{2/3}}{4} + (|\mathcal{W}_2| - t)\kappa n^{2/3} &\geq \left(\frac{1}{2} - \delta - 3n^{-1/3}\right)|\mathcal{W}_2| \cdot \frac{1}{4}n^{2/3} \\ \iff t(1 - 4\kappa) &\geq \left(\frac{1}{2} - \delta - 4\kappa - 3n^{-1/3}\right)|\mathcal{W}_2| \\ \iff t &\geq \left(\frac{1}{2} - \delta - 4\kappa - 3n^{-1/3}\right) \frac{|\mathcal{W}_2|}{1 - 4\kappa}. \quad \blacktriangleleft \end{aligned}$$

Now note that for every tile $W \in \mathcal{W}$ for which the row $M[\{W\}, V_2' \cup V_3']$ has weight at least $\kappa n^{2/3}$ we have that $d^+(v, F(W)) \geq \kappa n^{2/3}$, therefore by Lemma 12 the set $N^+(v) \cap F(W)$ $(1 - \kappa)$ -covers W . In other words, v controls at least $(1 - \kappa)n^{2/3}$ vertices in W .

Our goal is now to lower-bound the total number of such tiles, hence let s denote the total number of rows in $M[\mathcal{W}, V_2' \cup V_3']$ with weight at least $\kappa n^{2/3}$. By the previous two observations and by plugging in the concrete values of $\delta = \frac{2}{17}$ and $\kappa = \frac{1}{4000}$, we have that

$$\begin{aligned} s &\geq |\mathcal{W}_1| + \left(\frac{1}{2} - \delta - 4\kappa - 3n^{-1/3}\right) \frac{|\mathcal{W}_2|}{1 - 4\kappa} \\ &= |\mathcal{W}_1| + \left(\frac{6483}{17000} - 3n^{-1/3}\right) |\mathcal{W}_2| \frac{1000}{999}. \end{aligned}$$

Again we are aiming to prove a lower-bound, thus we assume that \mathcal{W}_1 is as small as possible. By Lemma 19, this means that

$$\begin{aligned} |\mathcal{W}_1| &= \left(\frac{1}{2} - \delta - \kappa\right)n^{1/3} - 2 = \frac{25983}{68000}n^{1/3} - 2 \quad \text{and} \\ |\mathcal{W}_2| &= \left(\frac{1}{2} + \delta + \kappa\right)n^{1/3} = \frac{42017}{68000}n^{1/3}. \end{aligned}$$

Plugging in the sizes of $\mathcal{W}_1, \mathcal{W}_2$ we obtain

$$\begin{aligned} s &\geq |\mathcal{W}_1| + \left(\frac{6483}{17000} - 3n^{-1/3}\right) |\mathcal{W}_2| \frac{1000}{999} \\ &\geq \frac{25983}{68000}n^{1/3} + \left(\frac{6483}{17000} - 3n^{-1/3}\right) \frac{42017}{67932}n^{1/3} - 2 \\ &\geq \frac{475777}{769896}n^{1/3} - 4. \end{aligned}$$

Since v controls a $(1 - \kappa) = \frac{3999}{4000}$ fraction of each tile counted by s and each tile has a size of $n^{2/3}$, we finally have the following lower bound on the second out-neighbourhood of v :

$$\begin{aligned} |N^{++}[v]| &\geq \frac{3999}{4000}sn^{2/3} \geq \frac{3999}{4000} \cdot \frac{475777}{769896}n - 4n^{2/3} \\ &= 0.61782\dots n - 4n^{2/3}. \end{aligned}$$

This value lies, for large enough n , above our target value of $(\frac{1}{2} + \delta)n = 0.61764\dots n$. \blacktriangleleft

5 Conclusion

We have shown how the usage of a *template-graph* helped us devise a seeker strategy that reveals a $(\frac{1}{2} + \frac{2}{17})$ -king in a tournament using $\tilde{O}(n^{4/3})$ queries, shedding light on a long-standing open problem. Our approach begins with a non-adaptive querying strategy based on what we called a *template graph*, which then helps to guide the seeker to identify a small set of queries which necessarily lead to the discovery of a $(\frac{1}{2} + \frac{2}{17})$ -king.

Naturally, we ask whether it is possible to find an improved strategy which reveals a $(\frac{1}{2} + \delta)$ -king with δ substantially larger than $\frac{2}{17}$ using a similar amount of queries.

References

- 1 Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and selection with imprecise comparisons. In *International Colloquium on Automata, Languages, and Programming*, pages 37–48. Springer, 2009.
- 2 Ramachandran Balasubramanian, Venkatesh Raman, and G Srinivasaragavan. Finding scores in tournaments. *Journal of Algorithms*, 24(2):380–394, 1997.
- 3 Arindam Biswas, Varunkumar Jayapaul, Venkatesh Raman, and Srinivasa Rao Satti. The complexity of finding (approximate sized) distance-d dominating set in tournaments. In Mingyu Xiao and Frances Rosamond, editors, *Frontiers in Algorithmics*, pages 22–33, Cham, 2017. Springer International Publishing.
- 4 Palash Dey. Query complexity of tournament solutions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.
- 5 Dishant Goyal, Varunkumar Jayapaul, and Venkatesh Raman. Elusiveness of finding degrees. *Discrete Applied Mathematics*, 286:128–139, 2020.
- 6 HG Landau. On dominance relations and the structure of animal societies: III the condition for a score structure. *The bulletin of mathematical biophysics*, 15(2):143–148, 1953.
- 7 John W Moon. *Topics on tournaments in graph theory*. Courier Dover Publications, 2015.
- 8 Jian Shen, Li Sheng, and Jie Wu. Searching for sorted sequences of kings in tournaments. *SIAM J. Comput.*, 32(5):1201–1209, 2003. doi:10.1137/S0097539702410053.

Complexity of Fault Tolerant Query Complexity

Ramita Maharjan ✉

University of Memphis, TN, USA

Thomas Watson ✉

University of Memphis, TN, USA

Abstract

In the model of fault tolerant decision trees introduced by Kenyon and Yao, there is a known upper bound E on the total number of queries that may be faulty (i.e., get the wrong bit). We consider this computational problem: Given as input the truth table of a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ and a value of E , find the minimum possible height (worst-case number of queries) of any decision tree that computes f while tolerating up to E many faults. We design an algorithm for this problem that runs in time $\tilde{O}\left(\binom{n+E}{E} \cdot (2E+3)^n\right)$, which is polynomial in the size of the truth table when E is a constant. This generalizes a standard algorithm for the non-fault tolerant setting.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Fault, Tolerant, Query, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.26

Funding This work was supported by NSF grant CCF-1657377.

1 Introduction

In practice, the fields of program synthesis (for software) and logic synthesis (for hardware) are about automating the design of computational processes: Given an input-output specification, generate an efficient implementation that meets the specification.

Complexity theory’s version of synthesis is known as “meta-complexity” or “complexity of complexity”: Given the truth table of a boolean function f , compute f ’s complexity – the cost of an optimal algorithm for f in some model of computation. Upper and lower bounds are known on the complexity of computing the complexity of a function (given its truth table) for various models, including circuits [16, 3, 4, 2, 15, 13], formulas [6, 5, 13, 14], branching programs [10, 6, 19], communication protocols [18, 15, 12], and decision trees [11, 1, 19].

We focus on decision trees. A decision tree computes a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ by adaptively querying (reading) individual bits of the input. A decision tree’s height (cost) is the maximum over all inputs of the number of queries made on that input. The query complexity of f is the minimum height of any decision tree computing f . A standard algorithm due to [11, 1] inputs f ’s truth table and outputs f ’s query complexity in time $\tilde{O}(3^n)$, where \tilde{O} hides a poly(n) factor. In terms of the input size $N = 2^n$, this is polynomial time $\tilde{O}(N^{\log_2(3)})$. A simple extension of the algorithm outputs an optimal decision tree, not just its height. There is also a considerable amount of literature on properly learning decision trees given access to input-output pairs of f (e.g., see the recent papers [8, 7] and the references within). The “given a truth table” problem corresponds to the extreme case where *all* input-output pairs of f are available.

We consider fault tolerant decision trees, in which a query to an input bit may or may not get the bit’s actual value. This is motivated by scenarios where the input bits are the results of unreliable computations. Several models of this type have been introduced [17, 9, 20, 21]. We focus on the original model from [17], in which a decision tree is only required to output the correct bit when the total number of faults, over all queries to all variables, is at most some parameter E . Letting \tilde{O} hide a poly(n, E) factor, we prove:



© Ramita Maharjan and Thomas Watson;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 26; pp. 26:1–26:11



Leibniz International Proceedings in Informatics

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

► **Theorem 1.** *Given as input the truth table of a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ and an integer $E \geq 0$, the E -fault tolerant query complexity of f can be computed in time $\tilde{O}\left(\binom{n+E}{E} \cdot (2E+3)^n\right)$.*

When E is a constant, this running time is $\tilde{O}(N^{\log_2(2E+3)})$, which is polynomial in the size $N = 2^n$ of f 's truth table.

Our algorithm generalizes the algorithm from [11, 1] for the non-fault tolerant setting. The analysis of our algorithm is much more involved, largely because our set of dynamic programming subproblems has a complicated combinatorial structure. It is not straightforward to characterize or count the subproblems. This stems from the fact that – unlike in the non-fault tolerant setting – we cannot treat the input variables as independent of each other.

2 Decision trees

A *decision tree* on boolean variables x_1, x_2, \dots, x_n is a full binary tree where each internal node is labeled with a variable index $i \in [n] = \{1, 2, \dots, n\}$ and each leaf is labeled with an output bit. On an input $x \in \{0, 1\}^n$, a decision tree follows a root-to-leaf path: Upon reaching an internal node labeled i , the decision tree *queries* x_i and goes to the left child if $x_i = 0$ or to the right child if $x_i = 1$. The output is the label of the leaf reached. A decision tree computes $f: \{0, 1\}^n \rightarrow \{0, 1\}$ iff it outputs $f(x)$ for every input x . A decision tree's *height* is the number of edges on a longest root-to-leaf path, i.e., the maximum over all $x \in \{0, 1\}^n$ of the number of queries made on input x . The (deterministic) *query complexity* of f , denoted $D(f)$, is the minimum height of any decision tree that computes f . We have $D(f) \leq n$ for all $f: \{0, 1\}^n \rightarrow \{0, 1\}$ since a decision tree can make n queries to determine the entire input x and then output $f(x)$. There is no reason for a decision tree computing f to query the same variable more than once along a root-to-leaf path.

In the fault tolerant setting, a query to x_i may get either x_i 's actual value or the opposite bit – a *fault* or *mistake*. We allow a decision tree to query the same variable more than once along a root-to-leaf path, in which case some queries may get the actual value while others are faults. We consider the model introduced by [17] where $E \geq 0$ is a specified upper bound on the total number of faults over all variables (not per variable). A decision tree computes $f: \{0, 1\}^n \rightarrow \{0, 1\}$ while tolerating E faults iff it outputs $f(x)$ for every input x and every choice of at most E queries to be faults. More formally, for a node v , input x , and variable index i , define $\text{faults}(v, x, i)$ as the number of nodes along the root-to- v path – excluding v itself – that are labeled i but the path goes to the wrong child, i.e., left child but $x_i = 1$, or right child but $x_i = 0$. Define $\text{faults}(v, x) = \sum_{i=1}^n \text{faults}(v, x, i)$. A decision tree is *E -fault tolerant* for f iff for every input x and every leaf v with $\text{faults}(v, x) \leq E$, the label of v is $f(x)$. In other words, if we define $\text{consistent}_E(v) = \{x : \text{faults}(v, x) \leq E\}$ then for every leaf v , $f(x)$ must be the same for all $x \in \text{consistent}_E(v)$. The *E -fault tolerant query complexity* of f , denoted $D_E(f)$, is the minimum height of any E -fault tolerant decision tree for f . We have $D(f) = D_0(f) \leq D_1(f) \leq D_2(f) \leq \dots$. The following lemma does not seem to appear in the literature.

► **Lemma 2.** $D_E(f) \leq D(f) \cdot (E + 1) + E$ for all f and E .

Proof. Let T be a decision tree of height $D(f)$ computing f . We design T' , which is an E -fault tolerant decision tree of height at most $D(f) \cdot (E + 1) + E$ for f . The idea is that on any input, T' tracks T 's root-to-leaf path but for each variable T queries along the way, T' queries the variable repeatedly until ascertaining its actual value:

- Suppose (for convenience) T first queries x_1 . We have T' query x_1 until some bit has appeared $E + 1$ times. This bit is x_1 's actual value since otherwise x_1 would have $E + 1$ faults. Let e_1 be the number of faults on x_1 . The number of queries to x_1 is $E + 1 + e_1$.

- Suppose (for convenience) T next queries x_2 . We have T' query x_2 until some bit has appeared $E + 1 - e_1$ times. This bit is x_2 's actual value since otherwise x_1, x_2 would have $E + 1$ faults. Let e_2 be the number of faults on x_2 . The number of queries to x_2 is $E + 1 - e_1 + e_2$.

⋮

- Suppose (for convenience) T queries x_i in the i^{th} step. We have T' query x_i until some bit has appeared $E + 1 - e_1 - e_2 - \dots - e_{i-1}$ times. This bit is x_i 's actual value since otherwise x_1, \dots, x_i would have $E + 1$ faults. Let e_i be the number of faults on x_i . The number of queries to x_i is $E + 1 - e_1 - e_2 - \dots - e_{i-1} + e_i$.

⋮

Upon reaching a leaf of T , we have T' output the same bit. If at most E faults occur, then T' finds the leaf T would reach. Since T computes f , T' is E -fault tolerant for f . If T makes q queries on input x , then T' makes at most

$$\begin{aligned} \sum_{i=1}^q (E + 1 - (\sum_{j=1}^{i-1} e_j) + e_i) &= q \cdot (E + 1) - \sum_{i=1}^q (q - 1 - i) \cdot e_i \\ &\leq q \cdot (E + 1) + e_q \\ &\leq D(f) \cdot (E + 1) + E \end{aligned}$$

queries on input x , regardless of when the faults occur. ◀

Lemma 2 is tight for the And function on n bits:

► **Observation 3.** $D_E(\text{And}_n) = n \cdot (E + 1) + E$ for all n and E .

Proof. An adversary can respond 1 to all queries, until some variable x_i has been queried E times and each other variable has been queried at least $E + 1$ times, and then the adversary can start responding 0 to subsequent queries to x_i (and keep responding 1 to queries to other variables). After at most $n \cdot (E + 1) + E - 1$ queries, the current node v would have $\text{faults}(v, y) \leq E$ where y is the all 1s input (so $\text{And}_n(y) = 1$) and $\text{faults}(v, z) \leq E$ where z is all 1s except $z_i = 0$ (so $\text{And}_n(z) = 0$), where i is the index of a variable that has been queried as 1 at most E times and as 0 at most E times. Thus v cannot be a leaf since $y, z \in \text{consistent}_E(v)$ but $f(y) \neq f(z)$. That is, an E -fault tolerant decision tree for And_n must make at least $n \cdot (E + 1) + E$ queries in the worst case, otherwise it may output the wrong bit for either y or z . ◀

Lemma 2 is not tight for all functions: Let Tribes_n be the function that interprets x as a $\sqrt{n} \times \sqrt{n}$ array of bits and $\text{Tribes}_n(x) = 1$ iff there exists an all-1 row in x . Then $D(\text{Tribes}_n) = n$ but [17] proved that $D_E(\text{Tribes}_n) = O(n + \sqrt{n} \cdot E)$. More generally, [17] proved that $D_E(f) = O(n + C(f) \cdot E)$ for all f and E , where $C(f) = \max(\text{CNF width of } f, \text{DNF width of } f)$ is the certificate complexity of f .

3 Patterns and redundant queries

A *pattern* is a tuple $p = (p_1, p_2, \dots, p_n) = ((p_{1,0}, p_{1,1}), (p_{2,0}, p_{2,1}), \dots, (p_{n,0}, p_{n,1}))$ where for all $i \in [n]$ and $b \in \{0, 1\}$, $p_{i,b}$ is a nonnegative integer representing the number of queries to variable x_i that got value b . Each node v in a decision tree has an associated pattern p^v that records the results of the queries along the root-to- v path but does not indicate the order of those queries. If v is the root, then $p^v = ((0, 0), (0, 0), \dots, (0, 0))$ since no queries have happened.

26:4 Complexity of Fault Tolerant Query Complexity

For a non-fault tolerant decision tree, in which no variable is ever re-queried, a pattern is traditionally viewed as a *restriction* $r \in \{0, 1, *\}^n$ where $r_i = 0$ means $x_i = 0$ was queried, and $r_i = 1$ means $x_i = 1$ was queried, and $r_i = *$ means x_i remains unqueried. Compared to our more general notion of patterns, $r_i = 0$ corresponds to $p_i = (1, 0)$, and $r_i = 1$ corresponds to $p_i = (0, 1)$, and $r_i = *$ corresponds to $p_i = (0, 0)$.

The combinatorial structure is more subtle for fault tolerant decision trees. For example, suppose $n = 2$, $E = 1$, the current pattern is $((1, 0), (0, 1))$, and we query x_1 . If the query gets $x_1 = 0$, then the pattern becomes $((2, 0), (0, 1))$ and we know x_1 is actually 0 (otherwise x_1 would have two faults) but we do not know x_2 's actual value. If the query gets $x_1 = 1$, then the pattern becomes $((1, 1), (0, 1))$ and we know x_2 is actually 1 (otherwise x_1, x_2 would each have one fault) but we do not know x_1 's actual value. The point is that querying a variable might reveal more information about *other* variables.

For a pattern p , we define $\text{faults}(p, x, i) = p_{i, \bar{x}_i}$ and $\text{faults}(p, x) = \sum_{i=1}^n \text{faults}(p, x, i)$ and $\text{consistent}_E(p) = \{x : \text{faults}(p, x) \leq E\}$. Thus for any node v in a decision tree, we have $\text{faults}(v, x, i) = \text{faults}(p^v, x, i)$ and $\text{faults}(v, x) = \text{faults}(p^v, x)$ and $\text{consistent}_E(v) = \text{consistent}_E(p^v)$.

For a pattern p , querying x_i would be *redundant* (with respect to E) iff all $x \in \text{consistent}_E(p)$ have the same value of x_i . An internal node v in a decision tree is *redundant* (with respect to E) iff x_i is a redundant query given p^v , where i is the label of v . A decision tree with no redundant nodes is *sensible* (with respect to E).

► **Lemma 4.** *For every f and E , there exists a minimum-height E -fault tolerant decision tree for f that is sensible.*

Proof. Consider any minimum-height E -fault tolerant decision tree for f . We claim that if v is a redundant node labeled i , and b is the common value of x_i for $x \in \text{consistent}_E(p^v)$, then the decision tree will still be E -fault tolerant for f after we replace v with v 's child corresponding to query $x_i = b$ and discard the other child's subtree. Repeating this process to eliminate all redundant nodes yields a sensible decision tree that is no taller than the original.

To prove the claim, consider any leaf u in v 's b -subtree in the original decision tree, and let w be the modified decision tree's leaf corresponding to u . We show that $\text{consistent}_E(p^w) = \text{consistent}_E(p^u)$, which implies that the modified decision tree is still E -fault tolerant for f since $f(x)$ is the same for all $x \in \text{consistent}_E(p^w)$. Trivially, $\text{consistent}_E(p^w) \supseteq \text{consistent}_E(p^u)$ since p^w is the same as p^u except $p_{i,b}^w = p_{i,b}^u - 1$ and so $\text{faults}(p^w, x) \leq \text{faults}(p^u, x)$ for all x . To see that $\text{consistent}_E(p^w) \subseteq \text{consistent}_E(p^u)$, first note that $p_{j,c}^w \geq p_{j,c}^u$ for all $(j, c) \in [n] \times \{0, 1\}$, because $p_{i,b}^w = p_{i,b}^u - 1 \geq p_{i,b}^u$ and $p_{j,c}^w = p_{j,c}^u \geq p_{j,c}^u$ for all $(j, c) \neq (i, b)$. Thus $\text{consistent}_E(p^w) \subseteq \text{consistent}_E(p^u)$, so $x_i = b$ for all $x \in \text{consistent}_E(p^w)$. This implies that $\text{faults}(p^u, x) = \text{faults}(p^w, x) \leq E$ for all $x \in \text{consistent}_E(p^w)$. ◀

► **Lemma 5.** *x_i is a redundant query given pattern p iff $\max_b(p_{i,b}) + \sum_{j \neq i} \min_b(p_{j,b}) > E$.*

Proof. ◀: Assume $\max_b(p_{i,b}) + \sum_{j \neq i} \min_b(p_{j,b}) > E$. For all $x \in \text{consistent}_E(p)$, we must have $x_i = \arg \max_b(p_{i,b})$ since otherwise $\text{faults}(p, x, i) = \max_b(p_{i,b})$ and $\text{faults}(p, x, j) \geq \min_b(p_{j,b})$ for all $j \neq i$ and thus $\text{faults}(p, x) \geq \max_b(p_{i,b}) + \sum_{j \neq i} \min_b(p_{j,b}) > E$, which would mean $x \notin \text{consistent}_E(p)$.

⇒: Assume $\max_b(p_{i,b}) + \sum_{j \neq i} \min_b(p_{j,b}) \leq E$. Define $y, z \in \{0, 1\}^n$ as follows: $y_i = 0$ and $z_i = 1$ and $y_j = z_j = \arg \max_b(p_{j,b})$ for all $j \neq i$. Then $\text{faults}(p, y, i), \text{faults}(p, z, i) \leq \max_b(p_{i,b})$ and $\text{faults}(p, y, j) = \text{faults}(p, z, j) = \min_b(p_{j,b})$ for all $j \neq i$, and thus $\text{faults}(p, y), \text{faults}(p, z) \leq \max_b(p_{i,b}) + \sum_{j \neq i} \min_b(p_{j,b}) \leq E$, which means $y, z \in \text{consistent}_E(p)$. Thus x_i is not a redundant query given p . ◀

For a pattern p , $i \in [n]$, and $b \in \{0, 1\}$, define the pattern $p^{i,b}$ to be the same as p except $p_{i,b}^{i,b} = p_{i,b} + 1$. If an internal node v has pattern p and label i , then v 's children have patterns $p^{i,0}$ and $p^{i,1}$.

► **Observation 6.** For all p and i , $\text{consistent}_E(p) = \text{consistent}_E(p^{i,0}) \cup \text{consistent}_E(p^{i,1})$.

Proof. \supseteq : For all $b \in \{0, 1\}$, if $x \in \text{consistent}_E(p^{i,b})$ then $\text{faults}(p, x) \leq \text{faults}(p^{i,b}, x) \leq E$ and so $x \in \text{consistent}_E(p)$.

\subseteq : If $x \in \text{consistent}_E(p)$ then $\text{faults}(p^{i,x_i}, x) = \text{faults}(p, x) \leq E$ and so $x \in \text{consistent}_E(p^{i,x_i})$. ◀

By the way, $\text{consistent}_E(p^{i,0})$ and $\text{consistent}_E(p^{i,1})$ might not be disjoint.

4 Valid patterns

A pattern p is *valid* (with respect to E) iff $p = p^v$ for some node v in some sensible decision tree. A centerpiece of the analysis of our algorithm is a characterization of valid patterns. To state this, we define:

- $\text{block}(p, d) = \{i \in [n] : |p_{i,0} - p_{i,1}| = d\}$ for each integer $d \geq 0$.
- $\text{min-faults}(p, D) = \sum_{d \geq D} \sum_{i \in \text{block}(p,d)} \min_b(p_{i,b})$ for each integer $D \geq 0$.
- $\text{min-faults}(p) = \text{min-faults}(p, 0) = \sum_{i=1}^n \min_b(p_{i,b})$.

► **Lemma 7.** p is valid iff $\text{min-faults}(p) \leq E$ and $\text{min-faults}(p, D) \leq E - D + 1$ for each $D \geq 2$ such that $\text{block}(p, D) \neq \emptyset$.

Proof. We define some notation with respect to p : For each variable x_i , define $b_i = \arg \max_b(p_{i,b})$ (breaking the tie arbitrarily if $p_{i,0} = p_{i,1}$) and $d_i = |p_{i,0} - p_{i,1}| = p_{i,b_i} - p_{i,\bar{b}_i}$, so $i \in \text{block}(p, d_i)$.

\Leftarrow : Assume $\text{min-faults}(p) \leq E$ and $\text{min-faults}(p, D) \leq E - D + 1$ for each $D \geq 2$ such that $\text{block}(p, D) \neq \emptyset$. The latter actually holds for $D \geq 1$, not just for $D \geq 2$, because $\text{min-faults}(p) \leq E$ implies that $\text{min-faults}(p, 1) \leq \text{min-faults}(p) \leq E = E - 1 + 1$.

To show that p is valid, we exhibit a sensible decision tree consisting of a root-to- v path where $p^v = p$ and all nodes hanging off of this path are leaves (whose outputs are irrelevant). For each variable x_i in decreasing order of d_i (breaking ties arbitrarily), the path has p_{i,\bar{b}_i} many $x_i = \bar{b}_i$ queries followed by p_{i,b_i} many $x_i = b_i$ queries. By definition, this path ends at a node v with $p^v = p$.

We argue that no internal node is redundant. Consider an internal node u labeled i . First suppose $d_i = 0$. Then $\max_b(p_{i,b}^u) \leq \min_b(p_{i,b})$ and $\min_b(p_{j,b}^u) \leq \min_b(p_{j,b})$ for each $j \neq i$. Thus

$$\begin{aligned} \max_b(p_{i,b}^u) + \sum_{j \neq i} \min_b(p_{j,b}^u) &\leq \sum_{j=1}^n \min_b(p_{j,b}) \\ &= \text{min-faults}(p) \\ &\leq E \end{aligned}$$

so u is not redundant, by Lemma 5. Now suppose $d_i \geq 1$. Then $\text{block}(p, d_i) \neq \emptyset$ since $i \in \text{block}(p, d_i)$, so $\text{min-faults}(p, d_i) \leq E - d_i + 1$ by assumption. We have $\max_b(p_{i,b}^u) \leq \max_b(p_{i,b}) - 1 = \min_b(p_{i,b}) + d_i - 1$ because $d_i \geq 1$ and the last $x_i = b_i$ query happens either at u or farther down the path (since the $x_i = b_i$ queries happen after the $x_i = \bar{b}_i$ queries). We have $\min_b(p_{j,b}^u) \leq \min_b(p_{j,b})$ for each $j \neq i$, and $\min_b(p_{j,b}^u) = 0$ if $d_j < d_i$ because variables are queried in decreasing order of d_j . Thus

26:6 Complexity of Fault Tolerant Query Complexity

$$\begin{aligned}
\max_b(p_{i,b}^u) + \sum_{j \neq i} \min_b(p_{j,b}^u) &\leq d_i - 1 + \sum_{d \geq d_i} \sum_{j \in \text{block}(p,d)} \min_b(p_{j,b}) \\
&= d_i - 1 + \text{min-faults}(p, d_i) \\
&\leq E
\end{aligned}$$

so u is not redundant, by Lemma 5.

\Rightarrow : Assume p is valid, and let v be a node with $p^v = p$ in some sensible decision tree.

To see that $\text{min-faults}(p) \leq E$, consider v 's parent u . (Or if v is the root, then this holds trivially.) We have $\min_b(p_{i,b}) \leq \max_b(p_{i,b}^u)$ where i is the label of u , and $\min_b(p_{j,b}) = \min_b(p_{j,b}^u)$ for each $j \neq i$. Thus

$$\begin{aligned}
\text{min-faults}(p) &= \sum_{j=1}^n \min_b(p_{j,b}) \\
&\leq \max_b(p_{i,b}^u) + \sum_{j \neq i} \min_b(p_{j,b}^u) \\
&\leq E
\end{aligned}$$

by Lemma 5, since u is not redundant.

To see that $\text{min-faults}(p, D) \leq E - D + 1$ for each $D \geq 2$ such that $\text{block}(p, D) \neq \emptyset$, consider the lowest node u on the root-to- v path (excluding v itself) such that u 's label i satisfies $d_i \geq D$. Such u exists since $D \geq 1$ and $\text{block}(p, D) \neq \emptyset$. Let w be u 's child on the root-to- v path (possibly $w = v$). Since u is the lowest such node, for each j with $d_j \geq D$ we have $p_{j,b}^w = p_{j,b}$ for both b and thus $\min_b(p_{j,b}^w) = \min_b(p_{j,b})$. We have $\min_b(p_{i,b}^w) = \max_b(p_{i,b}^w) - d_i \leq \max_b(p_{i,b}^u) - d_i + 1$ and $\min_b(p_{j,b}^w) = \min_b(p_{j,b}^u)$ for each $j \neq i$. Thus

$$\begin{aligned}
\text{min-faults}(p, D) &= \sum_{d \geq D} \sum_{j \in \text{block}(p,d)} \min_b(p_{j,b}) \\
&= \sum_{d \geq D} \sum_{j \in \text{block}(p,d)} \min_b(p_{j,b}^w) \\
&\leq \sum_{j=1}^n \min_b(p_{j,b}^w) \\
&\leq \max_b(p_{i,b}^u) - d_i + 1 + \sum_{j \neq i} \min_b(p_{j,b}^u) \\
&\leq E - d_i + 1 \\
&\leq E - D + 1
\end{aligned}$$

by Lemma 5, since u is not redundant. ◀

► **Corollary 8.** *If p is valid then $\text{consistent}_E(p) \neq \emptyset$.*

Proof. $x \in \text{consistent}_E(p)$ where each $x_i = \arg \max_b(p_{i,b})$ since $\text{faults}(p, x) = \text{min-faults}(p) \leq E$. ◀

► **Lemma 9.** *There are at most $\binom{n+E}{E} \cdot (2E+3)^n$ valid patterns.*

Proof. Choosing $p_{i,0}$ and $p_{i,1}$ is equivalent to choosing $\min_b(p_{i,b})$ and $p_{i,0} - p_{i,1}$ (no absolute value).

For valid p , there are $\binom{n+E}{E}$ possibilities for $(\min_b(p_{1,b}), \dots, \min_b(p_{n,b}))$ since $\text{min-faults}(p) \leq E$ by Lemma 7.

For valid p , we have $\text{block}(p, D) = \emptyset$ for all $D \geq E + 2$ since otherwise we would have the contradiction $0 \leq \text{min-faults}(p, D) \leq E - D + 1 < 0$ by Lemma 7. (Intuitively, a sensible decision tree would never re-query x_i after $\max_b(p_{i,b}) = E + 1$ since that would

be redundant by Lemma 5.) Thus for each i , there are $2E + 3$ possibilities for $p_{i,0} - p_{i,1}$, namely $E + 1, E, \dots, 0, \dots, -E, -E - 1$. Hence there are $(2E + 3)^n$ possibilities for $(p_{1,0} - p_{1,1}, \dots, p_{n,0} - p_{n,1})$.

There are at most $\binom{n+E}{E} \cdot (2E + 3)^n$ ways to form a valid p by choosing $(\min_b(p_{1,b}), \dots, \min_b(p_{n,b}))$ and $(p_{1,0} - p_{1,1}, \dots, p_{n,0} - p_{n,1})$. ◀

As a sanity check, when $E = 0$, Lemma 7 implies that p is valid iff $\text{min-faults}(p) = 0$ and $\text{block}(p, D) = \emptyset$ for all $D \geq 2$, which happens iff $p_i \in \{(1, 0), (0, 1), (0, 0)\}$ for each i . There are 3^n such patterns (corresponding to the restrictions in $\{0, 1, *\}^n$), which agrees with Lemma 9 when $E = 0$.

Lemma 9 generally overcounts the number of valid patterns because it ignores the constraints for $D \in \{2, \dots, E + 1\}$ in Lemma 7. The “minimum” and “difference” tuples are not independent. But Lemma 9 does not overcount by a lot: There are $\binom{n+E}{E}$ valid patterns with $p_{i,0} - p_{i,1} = 0$ for all i , and there are $(2E + 3)^n$ valid patterns with $\min_b(p_{i,b}) = 0$ for all i . Thus there are at least

$$\max\left(\binom{n+E}{E}, (2E + 3)^n\right) \geq \sqrt{\binom{n+E}{E} \cdot (2E + 3)^n}$$

valid patterns, so Lemma 9 is at least quadratically tight.

When E is a constant, the number of valid patterns is $\Theta\left(\binom{n+E}{E} \cdot (2E + 3)^n\right) = \Theta(n^E \cdot (2E + 3)^n)$. This is because each of the $\binom{n+E}{E}$ possibilities for $(\min_b(p_{1,b}), \dots, \min_b(p_{n,b}))$ has 0s in at least $n - E$ coordinates and thus gives rise to at least $(2E + 3)^{n-E} = \Omega((2E + 3)^n)$ valid patterns with the nonzero values of $p_{i,0} - p_{i,1}$ distributed only among the coordinates i with $\min_b(p_{i,b}) = 0$.

We can say something a little stronger than the previous paragraph. Suppose $1 \leq E \leq n$. There are $\binom{n}{E}$ possibilities of $(\min_b(p_{1,b}), \dots, \min_b(p_{n,b}))$ where $\min_b(p_{i,b}) = 1$ for E coordinates i and $\min_b(p_{i,b}) = 0$ for $n - E$ coordinates i . For each of those possibilities: There are $(2E + 3)^{n-E}$ possibilities for the tuple of $p_{i,0} - p_{i,1}$ for all i with $\min_b(p_{i,b}) = 0$. By Stirling’s formula (where e is the base of the natural log) there are $E! \geq (E/e)^E$ permutations of $[E]$, which we view as possibilities for the tuple of $|p_{i,0} - p_{i,1}|$ for all i with $\min_b(p_{i,b}) = 1$. There are 2^E possibilities for the tuple of $\text{sign}(p_{i,0} - p_{i,1})$ for all i with $\min_b(p_{i,b}) = 1$. In total, we just described at least

$$\binom{n}{E} \cdot (2E + 3)^{n-E} \cdot (E/e)^E \cdot 2^E \tag{†}$$

patterns p that are each valid, since $\text{min-faults}(p) = E$, and $\text{min-faults}(p, D) = E - D + 1$ for each $2 \leq D \leq E$, and $\text{block}(p, D) = \emptyset$ for each $D > E$. Let us compare this to the upper bound of Lemma 9. Assuming $1 \leq E \leq n/2$, we have

$$\begin{aligned} \binom{n+E}{E} / \binom{n}{E} &= \frac{(n+E) \cdot (n+E-1) \cdots (n+1)}{n \cdot (n-1) \cdots (n-E+1)} \\ &= (1+E/n) \cdot (1+E/(n-1)) \cdots (1+E/(n-E+1)) \\ &\leq 2^E \end{aligned}$$

and

$$\frac{(2E + 3)^n}{(2E + 3)^{n-E} \cdot (E/e)^E \cdot 2^E} = \left(\frac{(2E + 3) \cdot e}{2E}\right)^E = ((1 + 3/(2E)) \cdot e)^E \leq 7^E.$$

So, the upper bound $\binom{n+E}{E} \cdot (2E + 3)^n$, divided by the lower bound (†), is at most 14^E . Thus when $E = O(\log n)$, Lemma 9 only overcounts the number of valid patterns by a fairly insignificant poly(n) factor. Computational experiments suggest that Lemma 9 is fairly tight for all E , but it remains open to prove this.

5 The algorithm: Proof of Theorem 1

We design a dynamic programming algorithm for the following computational problem: Given as input the truth table of a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ and an integer $E \geq 0$, output $D_E(f)$.

For each valid pattern p , we have a subproblem whose solution is a pair (h, ℓ) where h is the minimum height of any sensible decision tree that is E -fault tolerant for f assuming the root is defined to have pattern p , and ℓ is the root's label in one such decision tree. If $h = 0$ then the root is a leaf that outputs $\ell \in \{0, 1\}$. If $h > 0$ then the root is an internal node that queries x_ℓ where $\ell \in [n]$. We store the solution (h, ℓ) to subproblem p in a dictionary data structure opt where p is the key and (h, ℓ) is the value. We use memoized recursion, rather than traditional dynamic programming, because enumerating the valid patterns is not straightforward.

Figure 1 shows the algorithm. If p is valid then $\text{solve}(p)$ returns the solution (h, ℓ) , after computing it and storing it in $opt[p]$ if it was not already there. We assume f , n , E , and opt are globally accessible in calls to the solve subroutine. The loop iterates through variables that are not redundant queries given p , by Lemma 5. By induction, we may assume $\text{solve}(p^{i,0})$ and $\text{solve}(p^{i,1})$ return correct solutions, since $p^{i,0}$ and $p^{i,1}$ are valid if x_i is not a redundant query given p .

Base cases are when the optimal height is 0, i.e., the root should be a leaf because $f(x)$ is the same for all $x \in \text{consistent}_E(p)$. One way to detect this would be to loop through all $x \in \{0, 1\}^n$, compute $\text{faults}(p, x)$, and compare $f(x)$ for all x with $\text{faults}(p, x) \leq E$, but that would incur $\tilde{O}(2^n)$ time overhead. Our algorithm detects base cases more efficiently. There are two kinds of base cases:

- $|\text{consistent}_E(p)| = 1$: By definition, every x_i is a redundant query given p , so we still have $h = \infty$ after the loop. Thus $\text{solve}(p)$ correctly computes the solution as $(0, f(x))$ where x is the unique element of $\text{consistent}_E(p)$. (There is no tie for $\arg \max_b(p_{i,b})$, since $\text{min-faults}(p) \leq E$ and $\text{min-faults}(p) + |p_{i,0} - p_{i,1}| > E$ implies $p_{i,0} \neq p_{i,1}$.)
- $|\text{consistent}_E(p)| > 1$ but $f(x)$ is the same for all $x \in \text{consistent}_E(p)$: By definition, at least one x_i is not a redundant query given p . When the loop reaches the first such index i , we have $h = \infty$ and $h_0 = h_1 = 0$ and $\ell_0 = \ell_1$ since $f(x)$ is the same for all $x \in \text{consistent}_E(p^{i,0}) \cup \text{consistent}_E(p^{i,1})$ by Observation 6 (\supseteq). Thus $\text{solve}(p)$ breaks out of the loop and correctly computes the solution as $(0, \ell_0)$ (skipping over “if $h = \infty$ ” after the loop since $h = 0$ now).

Non-base cases are when the optimal height is positive, i.e., the root should be an internal node because $f(x)$ is not the same for all $x \in \text{consistent}_E(p)$. Since we only consider sensible decision trees, the loop just tries every possible non-redundant query x_i the root could make, computes the height $1 + \max(h_0, h_1)$ of an optimal decision tree having i as the root label, lets h be the minimum of these heights, and lets ℓ be the index of the variable that achieves the minimum.¹ The condition “if $h = \infty$ and $h_0 = h_1 = 0$ and $\ell_0 = \ell_1$ ” is false since otherwise $f(x)$ would be the same for all $x \in \text{consistent}_E(p^{i,0}) \cup \text{consistent}_E(p^{i,1})$ and so p would be a base case by Observation 6 (\subseteq). Also, “if $h = \infty$ ” is false after the loop since at least one x_i is non-redundant given p , and h becomes finite during such an iteration. Thus for a non-base case, (h, ℓ) is correct at the end of $\text{solve}(p)$.

¹ If we wanted to minimize the number of leaves rather than the height, we could just change $1 + \max(h_0, h_1)$ to $h_0 + h_1$ and let $h = 1$ for base cases.


```

complexity( $f, E$ ):
  initialize empty dictionary  $opt$ 
   $(h, \ell) \leftarrow \text{solve}(((0, 0), \dots, (0, 0)))$ 
  return  $h$ 

solve( $p$ ):
  if  $opt$  contains key  $p$ : return  $opt[p]$ 
   $h \leftarrow \infty$ 
  for  $i \in [n]$ :
    if  $\text{min-faults}(p) + |p_{i,0} - p_{i,1}| \leq E$ :
       $(h_0, \ell_0) \leftarrow \text{solve}(p^{i,0})$ 
       $(h_1, \ell_1) \leftarrow \text{solve}(p^{i,1})$ 
      if  $h = \infty$  and  $h_0 = h_1 = 0$  and  $\ell_0 = \ell_1$ :
         $(h, \ell) \leftarrow (0, \ell_0)$  and break out of loop
      if  $h > 1 + \max(h_0, h_1)$ :  $(h, \ell) \leftarrow (1 + \max(h_0, h_1), i)$ 
  if  $h = \infty$ :  $(h, \ell) \leftarrow (0, f(x))$  where  $x_i = \arg \max_b(p_{i,b})$  for each  $i \in [n]$ 
   $opt[p] \leftarrow (h, \ell)$ 
  return  $(h, \ell)$ 

```

■ **Figure 1** Algorithm to compute $D_E(f)$.

Finally, $\text{complexity}(f, E)$ returns the minimum height of any sensible E -fault tolerant decision tree for f , which equals $D_E(f)$ by Lemma 4.

Now we analyze the running time. A call to $\text{solve}(p)$ is *fresh* iff opt does not yet contain the key p . We charge the cost of a nonfresh call to the parent call. There are at most $\binom{n+E}{E} \cdot (2E+3)^n$ fresh calls by Lemma 9, and each fresh call takes time $\text{poly}(n, E) = \tilde{O}(1)$ (excluding any fresh recursive calls it makes), assuming the dictionary is implemented with a self-balancing search tree. Thus the running time of $\text{complexity}(f, E)$ is $\tilde{O}\left(\binom{n+E}{E} \cdot (2E+3)^n\right)$.

After running $\text{solve}(((0, 0), \dots, (0, 0)))$, we can construct a minimum-height sensible E -fault tolerant decision tree for f by straightforwardly retracing the optimal choices for the relevant subproblems (Figure 2). In the non-fault tolerant setting ($E = 0$), constructing the decision tree is faster than solving all the subproblems: $\tilde{O}(2^n)$ compared to $\tilde{O}(3^n)$. In contrast, in the fault tolerant setting, constructing the decision tree can be much slower than solving all the subproblems – potentially as slow as $\tilde{O}(2^{n \cdot (E+1) + E})$ by Lemma 2 – since a fault tolerant decision tree can be a sprawling behemoth with many duplicated subtrees.

Another minor extension handles partial functions $f: \{0, 1\}^n \rightarrow \{0, 1, ?\}$ where $f(x) = ?$ means that $f(x)$ is undefined and we do not care what a decision tree outputs on input x . A decision tree is E -fault tolerant for a partial function f iff for every leaf v , its label equals $f(x)$ for all $x \in \text{consistent}_E(v)$ such that $f(x) \neq ?$. To compute $D_E(f)$ for a partial function f , we can modify $\text{solve}(p)$ so that $opt[p] = (0, ?)$ means that $f(x) = ?$ for all $x \in \text{consistent}_E(p)$, and $opt[p] = (0, \ell)$ for $\ell \in \{0, 1\}$ means that there exists an $x \in \text{consistent}_E(p)$ such that $f(x) \neq ?$ and that $f(x) = \ell$ for all such x . If $h = \infty$ and $h_0 = h_1 = 0$ then: If $\ell_0 = \ell_1$ or $\ell_1 = ?$, we assign $(h, \ell) \leftarrow (0, \ell_0)$. Else if $\ell_0 = ?$, we assign $(h, \ell) \leftarrow (0, \ell_1)$. Else we do not assign (h, ℓ) and do not break out of the loop here.

```

construct-tree( $p$ ):
  ( $h, \ell$ )  $\leftarrow$   $\text{opt}[p]$ 
  create a node  $v$  labeled  $\ell$ 
  if  $h > 0$ :
     $v$ 's left child  $\leftarrow$   $\text{construct-tree}(p^{\ell,0})$ 
     $v$ 's right child  $\leftarrow$   $\text{construct-tree}(p^{\ell,1})$ 
  return  $v$ 

```

■ **Figure 2** Constructing an optimal decision tree.

6 Future directions

It is open to prove asymptotically tight bounds on the number of valid patterns for all n and E .

When E is super-constant, our algorithm's running time is super-polynomial in the input size. Can we prove a complexity lower bound on the problem for large E ? Even in the non-fault tolerant setting, can fine-grained complexity techniques show that the $\tilde{O}(N^{\log_2(3)})$ running time is optimal under standard assumptions?

It remains open to study the complexity of computing fault tolerant *randomized* query complexity. The algorithm from [1] for computing non-fault tolerant randomized query complexity does not seem to generalize to the fault tolerant setting.

It is also open to study the analogous question for other models of fault tolerant decision trees, such as those introduced in [9]. To the best of our knowledge, “complexity of complexity” has not been studied for models of fault tolerant circuits, formulas, branching programs, or communication protocols.

References

- 1 Scott Aaronson. Algorithms for boolean function query properties. *SIAM Journal on Computing*, 32(5):1140–1157, 2003. doi:10.1137/S0097539700379644.
- 2 Eric Allender. The new complexity landscape around circuit minimization. In *Proceedings of the 14th Conference on Language and Automata Theory and Applications (LATA)*, pages 3–16. Springer, 2020. doi:10.1007/978-3-030-40608-0_1.
- 3 Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. *Information and Computation*, 256:2–8, 2017. doi:10.1016/j.ic.2017.04.004.
- 4 Eric Allender, Joshua Grochow, Dieter van Melkebeek, Cristopher Moore, and Andrew Morgan. Minimum circuit size, graph isomorphism, and related problems. *SIAM Journal on Computing*, 47(4):1339–1372, 2018. doi:10.1137/17M1157970.
- 5 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing DNF formulas and AC_d^0 circuits given a truth table. In *Proceedings of the 21st Conference on Computational Complexity (CCC)*, pages 237–251. IEEE, 2006. doi:10.1109/CCC.2006.27.
- 6 Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. Derandomization and distinguishing complexity. In *Proceedings of the 18th Conference on Computational Complexity (CCC)*, pages 209–220. IEEE, 2003. doi:10.1109/CCC.2003.1214421.
- 7 Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Properly learning decision trees in almost polynomial time. In *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*, pages 920–929. IEEE, 2021. doi:10.1109/FOCS52979.2021.00093.

- 8 Nader Bshouty and Catherine Haddad-Zaknoon. Adaptive exact learning of decision trees from membership queries. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory (ALT)*, pages 207–234. PMLR, 2019.
- 9 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.
- 10 Steven Friedman and Kenneth Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39(5):710–713, 1990. doi:10.1109/12.53586.
- 11 David Guijarro, Víctor Lavín, and Vijay Raghavan. Exact learning when irrelevant variables abound. *Information Processing Letters*, 70(5):233–239, 1999. doi:10.1016/S0020-0190(99)00063-0.
- 12 Shuichi Hirahara, Rahul Ilango, and Bruno Loff. Hardness of constant-round communication complexity. In *Proceedings of the 36th Computational Complexity Conference (CCC)*, pages 31:1–31:30. Schloss Dagstuhl, 2021. doi:10.4230/LIPIcs.CCC.2021.31.
- 13 Rahul Ilango. Constant depth formula and partial function versions of MCSP are hard. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, pages 424–433. IEEE, 2020. doi:10.1109/FOCS46700.2020.00047.
- 14 Rahul Ilango. The minimum formula size problem is (ETH) hard. In *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*, pages 427–432. IEEE, 2021. doi:10.1109/FOCS52979.2021.00050.
- 15 Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. NP-hardness of circuit minimization for multi-output functions. In *Proceedings of the 35th Computational Complexity Conference (CCC)*, pages 22:1–22:36. Schloss Dagstuhl, 2020. doi:10.4230/LIPIcs.CCC.2020.22.
- 16 Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *Proceedings of the 32nd Symposium on Theory of Computing (STOC)*, pages 73–79, 2000. doi:10.1145/335305.335314.
- 17 Claire Kenyon and Andrew Yao. On evaluating boolean functions with unreliable tests. *International Journal of Foundations of Computer Science*, 1(1):1–10, 1990. doi:10.1142/S0129054190000023.
- 18 Eyal Kushilevitz and Enav Weinreb. On the complexity of communication complexity. In *Proceedings of the 41st Symposium on Theory of Computing (STOC)*, pages 465–474. ACM, 2009. doi:10.1145/1536414.1536479.
- 19 Netanel Raviv. Truth table minimization of computational models. Technical report, arXiv, 2013. arXiv:1306.3766.
- 20 Rüdiger Reischuk and Bernd Schmelz. Reliable computation with noisy circuits and decision trees – A general $n \log n$ lower bound. In *Proceedings of the 32nd Symposium on Foundations of Computer Science (FOCS)*, pages 602–611. IEEE, 1991. doi:10.1109/SFCS.1991.185425.
- 21 Mario Szegedy and Xiaomin Chen. Computing boolean functions from multiple faulty copies of input bits. *Theoretical Computer Science*, 321(1):149–170, 2004. doi:10.1016/j.tcs.2003.07.001.

Romeo and Juliet Meeting in Forest like Regions

Neeldhara Misra ✉ 🏠 

Indian Institute of Technology, Gandhinagar, India

Manas Mulpuri ✉

Indian Institute of Technology, Gandhinagar, India

Prafullkumar Tale ✉ 🏠

Indian Institute of Science Education and Research, Pune, India

Gaurav Viramgami ✉

Indian Institute of Technology, Gandhinagar, India

Abstract

The game of rendezvous with adversaries is a game on a graph played by two players: *Facilitator* and *Divider*. Facilitator has two agents and Divider has a team of $k \geq 1$ agents. While the initial positions of Facilitator's agents are fixed, Divider gets to select the initial positions of his agents. Then, they take turns to move their agents to adjacent vertices (or stay put) with Facilitator's goal to bring both her agents at same vertex and Divider's goal to prevent it. The computational question of interest is to determine if Facilitator has a winning strategy against Divider with k agents. Fomin, Golovach, and Thilikos [WG, 2021] introduced this game and proved that it is PSPACE-hard and co-W[2]-hard parameterized by the number of agents.

This hardness naturally motivates the structural parameterization of the problem. The authors proved that it admits an FPT algorithm when parameterized by the modular width and the number of allowed rounds. However, they left open the complexity of the problem from the perspective of other structural parameters. In particular, they explicitly asked whether the problem admits an FPT or XP-algorithm with respect to the treewidth of the input graph. We answer this question in the negative and show that RENDEZVOUS is co-NP-hard even for graphs of constant treewidth. Further, we show that the problem is co-W[1]-hard when parameterized by the feedback vertex set number and the number of agents, and is unlikely to admit a polynomial kernel when parameterized by the vertex cover number and the number of agents. Complementing these hardness results, we show that the RENDEZVOUS is FPT when parameterized by both the vertex cover number and the solution size. Finally, for graphs of treewidth at most two and grids, we show that the problem can be solved in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Design and analysis of algorithms

Keywords and phrases Games on Graphs, Dynamic Separators, W[1]-hardness, Structural Parameterization, Treewidth

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.27

Related Version *Full Version*: <https://arxiv.org/abs/2210.02582> [6]

Funding *Neeldhara Misra*: The author is grateful for support from DST-SERB and IIT Gandhinagar. This work was partially supported by the ECR grant ECR/2018/002967.

Prafullkumar Tale: Part of the work was carried out when the author was a Post-Doctoral Researcher at CISA Helmholtz Center for Information Security, Germany, supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

Acknowledgements We are grateful for feedback from anonymous reviewers.



© Neeldhara Misra, Manas Mulpuri, Prafullkumar Tale, and Gaurav Viramgami; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 27; pp. 27:1–27:22



Leibniz International Proceedings in Informatics

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

1 Introduction

The game of rendezvous with adversaries on a graph – RENDEZVOUS – is a natural dynamic version of the problem of finding a vertex cut between two vertices s and t introduced by Fomin, Golovach, and Thilikos [4]. The game is played on a finite undirected connected graph G by two players: *Facilitator* and *Divider*. Facilitator has two agents Romeo and Juliet that are initially placed in designated vertices s and t of G . Divider, on the other hand, has a team of $k \geq 1$ agents D_1, \dots, D_k that are initially placed in some vertices of $V(G) \setminus \{s, t\}$ chosen by him. We note that a single vertex can accommodate multiple agents of Divider.

Then the players make their moves by turn, starting with Facilitator. At every move, each player moves some of his/her agents to adjacent vertices or keeps them in their old positions. No agent can be moved to a vertex that is currently occupied by adversary’s agents. Both players have complete information about G and the positions of all the agents. Facilitator aims to ensure that Romeo and Juliet meet; that is, they are in the same vertex. The task of Divider is to prevent the rendezvous of Romeo and Juliet by maintaining D_1, \dots, D_k in positions that block the possibility to meet. Facilitator wins if Romeo and Juliet meet, and Divider wins if they succeed in preventing the meeting of Romeo and Juliet forever. This setup naturally leads to the following computational question.

RENDEZVOUS

Input: A graph G with two given vertices s and t , and a positive integer k .

Question: Can Facilitator win on G starting from s and t against Divider with k agents?

We will often refer to k , the number of agents employed by Divider to keep Romeo and Juliet separated, as the “solution size” for this problem.

Known Results

Fomin, Golovach, and Thilikos [4] initiated an extensive study of the computational complexity of RENDEZVOUS. They concluded that the problem is PSPACE-hard and co-W[2]-hard¹ when parameterized by the number of Divider’s agents, while also demonstrating an $|V(G)|^{\mathcal{O}(k)}$ algorithm based on backtracking stages over the game arena. They also show that the problem admits polynomial time algorithms on chordal graphs and P_5 -free graphs. A related problem considered is RENDEZVOUS IN TIME, which asks if Facilitator can force a win in at most τ steps. It turns out that RENDEZVOUS IN TIME is co-NP-complete even for $\tau = 2$ and is FPT when parameterized by τ and the neighborhood diversity of the graph. The latter is an ILP-based approach and uses the fact that INTEGER LINEAR PROGRAMMING FEASIBILITY is FPT in the number of variables. We refer readers to [4], and references within, for more related problems.

The smallest number of agents that Divider needs to use to win on a graph G is called the “dynamic” separation number of G . We denote this by $d_G(s, t)$. Note that if s and t are adjacent or $s = t$, then $d_G(s, t) := +\infty$. The “static” separation number between s and

¹ We refer the reader to Appendix A.1 for the definitions of these complexity classes.

t , the original positions of Facilitator’s agents, is simply the smallest size of a (s, t) -vertex cut, i.e, a subset of vertices whose removal disconnects s and t . We use $\lambda_G(s, t)$ to denote the minimum size of a vertex (s, t) -separator in G . It is clear that $d_G(s, t) \leq \lambda_G(s, t)$, since positioning $\lambda_G(s, t)$ many guards on the vertices of a (s, t) -vertex allows Divider to win the game right away. It turns out that $d_G(s, t) = 1$ if and only if $\lambda_G(s, t) = 1$. However, there are examples of graphs where $d_G(s, t)$ is arbitrarily smaller than $\lambda_G(s, t)$ [4]. The results in [4] for chordal graphs and P_5 -free graphs are based on the fact that in these graphs, it turns out that $d_G(s, t) = \lambda_G(s, t)$.

Our Contributions

Given that the problem is hard in the solution size, often regarded the “standard” parameter, a natural approach is to turn to structural parameters of the input graph. One of the most popular structural parameters in the context of graphs is treewidth, which is a measure of how “tree-like” a graph is. XP and FPT algorithms parameterized by treewidth are natural generalizations of tractability on trees. Indeed, RENDEZVOUS is easy to solve on trees because $\lambda_G(s, t) = 1$ for any distinct s and t when G is a tree and $st \notin E(G)$. The complexity of RENDEZVOUS parameterized by treewidth, however, is wide open – in particular it is not even known if the problem is in XP parameterized by treewidth.

Interestingly, it was pointed out in [4] that if the initial positions s and t are not in the same bag of a tree decomposition of width w , then the upper bound for the dynamic separation number by $\lambda_G(s, t)$ together with the XP algorithm for the standard parameter can be employed to solve the problem in $n^{O(w)}$ time. Thus, the question that was left open by Fomin, Golovach, and Thilikos was if the problem can be solved in the same time if s and t are in the same bag. Our first contribution is to answer this question in the negative by showing that RENDEZVOUS is in fact co-NP-hard even for graphs of *constant* treewidth. In fact, we show more:

► **Theorem 1.** RENDEZVOUS is co-NP-hard even when restricted to:

- graphs whose feedback vertex set number is at most 14, or
- graphs whose pathwidth is at most 16.

In particular, RENDEZVOUS is co-para-NP-hard parameterized by treewidth.

We obtain this hardness by a non-trivial reduction from the 3-DIMENSIONAL MATCHING problem. In the backdrop of this somewhat surprising result, we are motivated to pursue the question of the complexity of RENDEZVOUS for larger parameters. It turns out that even augmenting the feedback vertex set number or the pathwidth with the solution size is not enough. Specifically, we show that the problem is unlikely to admit an FPT-algorithm even when parameterized by these combined parameters.

► **Theorem 2.** RENDEZVOUS is co-W[1]-hard when parameterized by:

- the feedback vertex set number and the solution size, or
- the pathwidth and the solution size.

This result is shown by a parameter preserving reduction from the (MONOTONE) NAE-INTEGERS-3-SAT problem, which was shown to be W[1]-hard when parameterized by the number of variables by Bringmann et al. [1]. Note that with this, we have a reasonably complete understanding of RENDEZVOUS in the combined parameter. Indeed, recall that the problem is co-W[1]-hard and XP parameterized by the solution size alone, and co-para-NP-hard parameterized by the feedback vertex set number alone as shown above.

Given the above hardness, we consider RENDEZVOUS parameterized by the vertex cover number, a larger parameter compared to both the feedback vertex set number and pathwidth. The status of RENDEZVOUS with respect to the vertex cover parameterization was also left

open in [4]. We see that the problem admits a natural exponential kernel in this parameter when combined with the solution size, and is hence FPT in the combined parameter; however this kernel cannot be improved to a polynomial kernel under standard complexity-theoretic assumptions.

► **Theorem 3.** *RENDEZVOUS is FPT when parameterized by the vertex cover number of the input graph and the solution size. Moreover, the problem does not admit a polynomial kernel when parameterized by the vertex cover number and the solution size unless $\text{NP} \subseteq \text{co-NP}/\text{poly}$.*

We briefly describe the intuition for the exponential kernel with respect to the vertex cover number. Suppose the graph G has a vertex cover X , where $|X| \leq \ell$, and, one may assume, without loss of generality, that $s, t \in X$. Further, for any $Y \subseteq X$, let I_Y denote the set of all vertices in $G \setminus X$ whose neighborhood in X is exactly Y . It is not hard to see that if $|I_Y| > k$, then one might as well “curtail” the set to $k + 1$ vertices without changing the instance. This leads to an exponential kernel in the combined parameter. It is also true that k is bounded, without loss of generality, by ℓ and the size of the common neighborhood of s and t to begin with; however, it is unclear if k can always be bounded by some function of the vertex cover alone. The kernelization lower bound follows from observing the structure of the reduced instance in the reduction used in [4] to prove that problem is $\text{co-W}[2]$ -hard when parameterized by the solution size.

Finally, we present polynomial time algorithms on two restricted cases.

► **Theorem 4.** *RENDEZVOUS can be solved in polynomial time on the classes of treewidth at most two graphs and grids.*

Recall that the polynomial time algorithm on the classes of trees, chordal graphs, and P_5 -free graphs is obtained by proving that the size of dynamic separator is same as that of separator. In case of grids, we present a winning strategy for Divider for any non-trivial instances. This makes grids unique graph class in which the problem admits polynomial time algorithm even when dynamic separator can be smaller than separator.

Organization of the paper. Due to lack of space, we defer several technical proofs, the polynomial time algorithms, and the preliminaries to the full version [6]. We describe the main intuitions for the proofs of Theorem 1 Section Section 2 We present a formal proof of Theorem 2 in Section 3. The proof of Theorem 3 partially discussed in Section 4. The hardness result can be found in [6].

2 co-para-NP-hardness Parameterized by FVS and Pathwidth

In this section, we prove that RENDEZVOUS is paraNP-hard when parameterized by the feedback vertex set number and the pathwidth of the input graph. To do that, we present a parameter preserving reduction from the 3-DIMENSIONAL MATCHING problem, which is known to be NP-hard [5, SP 1]. For notational convenience, we work with the following definition of the problem. An input consists of a universe $\mathcal{U} = \{\alpha, \beta, \gamma\} \times [n]$, a family $\mathcal{F} = \{A_1, A_2, \dots, A_m\}$ of subsets of \mathcal{U} such that for every $j \in [m]$, set $A_j = \{(\alpha, a_1), (\beta, b_1), (\gamma, c_1)\}$ for some $a_1, b_1, c_1 \in [n]$. The goal is to find a subset $\mathcal{F}' \subseteq \mathcal{F}$ that covers \mathcal{U} (and contains exactly n sets).

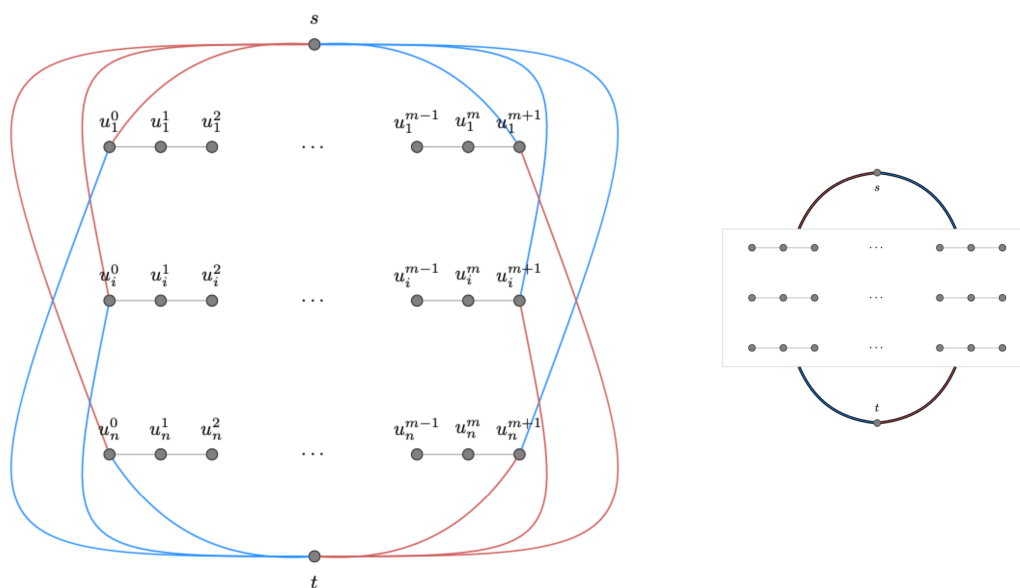
Reduction

The reduction takes as input an instance $(\mathcal{U}, \mathcal{F})$ of 3-DIMENSIONAL MATCHING and returns an instance (G, s, t, k) of RENDEZVOUS. It defines $M = n^2 + m^2$ where $n = |\mathcal{U}|/3$ and $m = |\mathcal{F}|$. We construct the graph G as follows: (c.f. Figures 1–4).

The Base Gadget. It starts by adding special vertices s and t and two more vertices g_1 and g_2 , and makes them common neighbours of s and t . We use $P[u, v, d]$ to denote a simple path from u to v that contains d many internal vertices.

- For every $i \in [n]^2$, it adds the following simple paths:
 - $P[u_i^0, u_i^{m+1}, m]$,
 - $P[s, u_i^0, m]$, $P[s, u_i^{m+1}, m]$, $P[t, u_i^0, m]$, and $P[t, u_i^{m+1}, m]$.

See Figure 1 for an illustration.



■ **Figure 1** (Left) The base gadget except the guard vertices g_1 and g_2 (which are not shown for clarity). Each red and blue path has m internal vertices. (Right) Schematic representation.

Encoding Elements. The reduction constructs a symmetric graph to encode elements in \mathcal{U} and has “left-side” and “right-side.” It starts by adding vertices $\{\alpha^\ell, \beta^\ell, \gamma^\ell\}$ and $\{\alpha^r, \beta^r, \gamma^r\}$.

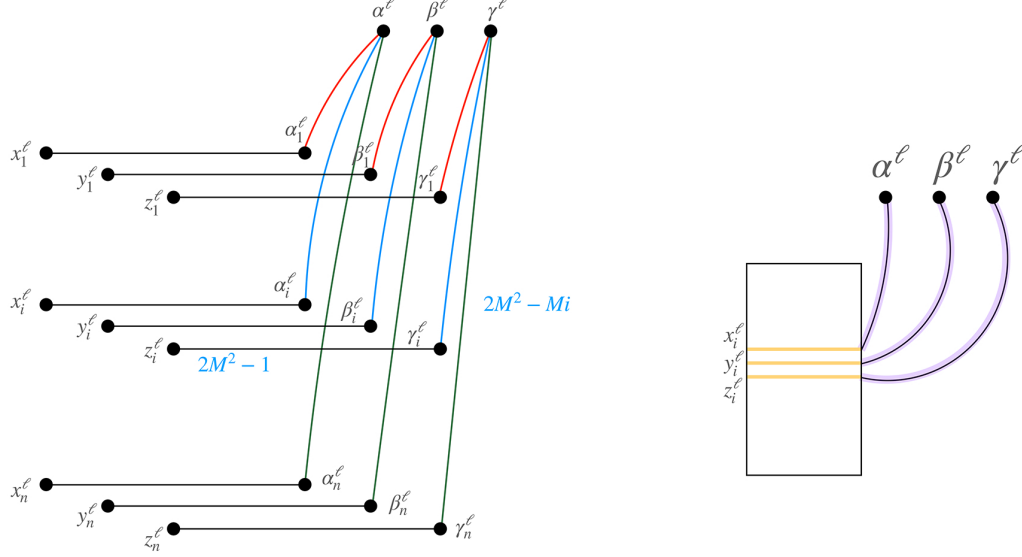
- For every $i \in [n]$, it adds six vertices in $\{\alpha_i^\ell, \beta_i^\ell, \gamma_i^\ell\} \cup \{\alpha_i^r, \beta_i^r, \gamma_i^r\}$, and the following simple paths:
 - $P[\alpha^\ell, \alpha_i^\ell, M^2 - M \cdot i]$, $P[\beta^\ell, \beta_i^\ell, M^2 - M \cdot i]$, $P[\gamma^\ell, \gamma_i^\ell, M^2 - M \cdot i]$,
 - $P[\alpha^r, \alpha_i^r, M^2 + M \cdot i]$, $P[\beta^r, \beta_i^r, M^2 + M \cdot i]$, and $P[\gamma^r, \gamma_i^r, M^2 + M \cdot i]$.

Note that the number of internal vertices in paths from α^ℓ to α_i^ℓ and from α^r to α_i^r , and similar such pairs, are different and depend on i .

² We use i as well as a_1, b_1, c_1 as running variables in set $[n]$. We reserve later types of variables for the integer part of elements in sets \mathcal{F} .

27:6 Romeo and Juliet Meeting in Forest like Regions

- For every $i \in [n]$, it adds six vertices $\{x_i^\ell, y_i^\ell, z_i^\ell\} \cup \{x_i^r, y_i^r, z_i^r\}$, and the following simple paths:
 - $P[x_i^\ell, \alpha_i^\ell, 2M^2 - 1]$, $P[y_i^\ell, \beta_i^\ell, 2M^2 - 1]$, $P[z_i^\ell, \gamma_i^\ell, 2M^2 - 1]$,
 - $P[x_i^r, \alpha_i^r, 2M^2 - 1]$, $P[y_i^r, \beta_i^r, 2M^2 - 1]$, and $P[z_i^r, \gamma_i^r, 2M^2 - 1]$.
- See Figure 2 for an illustration.



■ **Figure 2** (Left) The left side of the gadget is added to encode elements in \mathcal{U} . The number of internal vertices in each red, blue, and green path depends on i . The number of internal vertices in each yellow shaded path is $2M^2 - 1$. (Right) Schematic representation of the gadget.

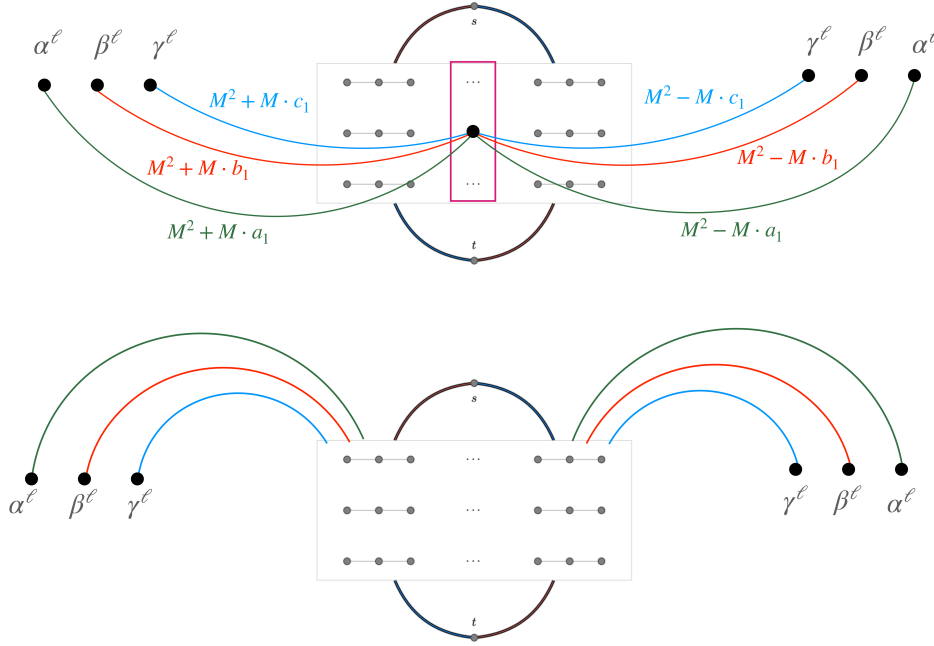
Encoding sets. The reduction adds simple paths to encode sets. Consider set A_j for some $j \in [m]$. Suppose the internal vertices of $P[u_i^0, u_i^{m+1}, m]$ are denoted by u_i^j for every $j \in [m]$, and u_i^0 is adjacent with u_i^1 and u_i^{m+1} is adjacent with u_i^m . All the vertices in j^{th} 'column' corresponds to set A_j . This, however, is not an encoding of set A_j as it does not provide any information about its elements. By the definition of the problem, set A_j has an element of the form (α, a_1) . To encode this element, it adds $2n$ many paths connecting j^{th} column to α^ℓ and to α^r . The number of internal vertices in these paths depends on a_1 . We encode the remaining two elements in A_j similarly. We formalise this construction as follows:

- For every $j \in [m]$, suppose $A_j = \{(\alpha, a_1), (\beta, b_1), (\gamma, c_1)\}$. Then, for every $i \in [n]$, the reduction adds the following six simple paths:
 - $P[\alpha^\ell, u_i^j, M^2 + M \cdot a_1]$, $P[\beta^\ell, u_i^j, M^2 + M \cdot b_1]$, $P[\gamma^\ell, u_i^j, M^2 + M \cdot c_1]$,
 - $P[\alpha^r, u_i^j, M^2 - M \cdot a_1]$, $P[\beta^r, u_i^j, M^2 - M \cdot b_1]$, and $P[\gamma^r, u_i^j, M^2 - M \cdot c_1]$.

See Figure 3 for an illustration.

Critical vertices and connecting paths. In the last phase of the reduction, it adds critical vertices and connect them to $\{s, t\}$, and also to x -type, y -type, and z -type ends of paths added while encoding elements in \mathcal{U} .

- For the special vertex s , it adds critical vertices, say $s_\alpha^\ell, s_\beta^\ell$ and s_γ^ℓ , on the left side.
 - It adds $P[s, s_\alpha^\ell, 2M^2 + 1]$, $P[s, s_\beta^\ell, 2M^2 + 1]$, and $P[s, s_\gamma^\ell, 2M^2 + 1]$.
 - For every $i \in [n]$, it adds $P[s_\alpha^\ell, x_i^\ell, 2M^2]$, $P[s_\beta^\ell, y_i^\ell, 2M^2]$, and $P[s_\gamma^\ell, z_i^\ell, 2M^2]$.



■ **Figure 3** (Top) Vertices added to encode sets in \mathcal{F} . The number of internal vertices in the paths depend on elements in A_j and are denoted next to it. (Bottom) Schematic representation of the gadget used in subsequent figures.

It adds the other critical vertices and paths symmetrically. We present them for the sake of completeness.

For the special vertex s , it adds critical vertices, say s_α^r , s_β^r , and s_γ^r , on the right side.

- It adds $P[s, s_\alpha^r, 2M^2 + 1]$, $P[s, s_\beta^r, 2M^2 + 1]$, and $P[s, s_\gamma^r, 2M^2 + 1]$.
- For every $i \in [n]$, it adds $P[s_\alpha^r, x_i^r, 2M^2]$, $P[s_\beta^r, y_i^r, 2M^2]$, and $P[s_\gamma^r, z_i^r, 2M^2]$.

For the special vertex t , it adds critical vertices, say t_α^ℓ , t_β^ℓ , t_γ^ℓ , on the left side.

- It adds $P[t, t_\alpha^\ell, 2M^2 + 1]$, $P[t, t_\beta^\ell, 2M^2 + 1]$, and $P[t, t_\gamma^\ell, 2M^2 + 1]$.
- For every $i \in [n]$, it adds $P[t_\alpha^\ell, x_i^\ell, 2M^2]$, $P[t_\beta^\ell, y_i^\ell, 2M^2]$, $P[t_\gamma^\ell, z_i^\ell, 2M^2]$,

For the special vertex t , it adds critical vertices, say t_α^r , t_β^r , and t_γ^r , on the right side.

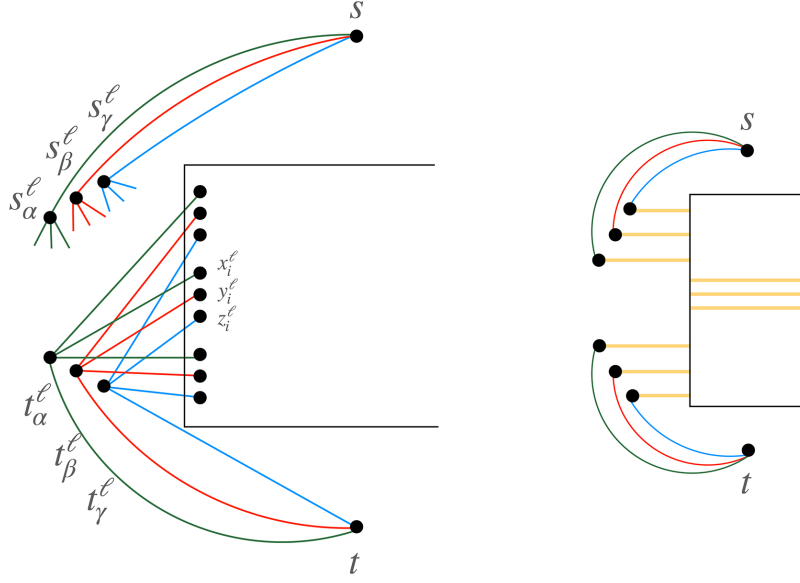
- It adds $P[t, t_\alpha^r, 2M^2 + 1]$, $P[t, t_\beta^r, 2M^2 + 1]$, and $P[t, t_\gamma^r, 2M^2 + 1]$.
- For every $i \in [n]$, it adds $P[t_\alpha^r, x_i^r, 2M^2]$, $P[t_\beta^r, y_i^r, 2M^2]$, and $P[t_\gamma^r, z_i^r, 2M^2]$.

This completes the construction of the graph G . The reduction sets $k = n + 2$ and returns (G, s, t, k) as the reduced instance of RENDEZVOUS.

Intuition for the correctness

We present an intuition for the correctness of the reduction in the reverse direction. In other words, we state how the initial positions of the Divider's agent correspond to sets and the elements they can cover. We start with determining the possible initial positions.

As g_1, g_2 are common neighbors of s and t , Divider needs to put two agents on g_1 and g_2 . For the remaining n agents, consider the paths $P[s, u_i^0, m]$ and $P[u_i^0, t, m]$ or $P[s, u_i^{m+1}, m]$ and $P[u_i^{m+1}, t, m]$ for every $i \in [n]$. Facilitator can move both Romeo and Juliet to u_i^0 or u_i^{m+1} in m steps. Hence, Divider needs to place remaining n agents at the positions that are at distance at most m simultaneously from u_i^0 and u_i^{m+1} . We ensure that he needs to place



■ **Figure 4** (Left) Critical vertices added by the reduction. The number of internal vertices in the paths are fixed ($2M^2 + 1$ or $2M^2$). (Right) Schematic representation of the gadget.

an agent on an internal vertex of $P[u_i^0, u_i^{m+1}, m]$ for every $i \in [n]$. This will correspond to selecting a set in \mathcal{F} in a solution. Formally, an agent at u_i^j for some $j \in [m]$ corresponds to selecting A_j in the cover of \mathcal{U} . As there are n 'rows', this will correspond to selecting n (different) sets from \mathcal{F} . Hence, the initial position of the Divider's agent will correspond to a collection of sets in \mathcal{F} .

Suppose for every $i \in [n]$, vertices in $\{x_i^l, x_i^r\}$ correspond to element $(\alpha, i) \in \mathcal{U}$. Similarly, vertices in $\{y_i^l, y_i^r\}$ correspond to (β, i) , and vertices in $\{z_i^l, z_i^r\}$ correspond to (γ, i) . We say (α, i) is covered if Divider can prevent Facilitator from moving both Romeo and Juliet at x_i^l as well as at x_i^r .

From the Facilitator's preservative, she has $6n$ possible meeting points of the above form. She can make these choices in two phases. In the first phase, she can decide to move Romeo towards one of the six vertices in $\{s_\alpha^l, s_\beta^l, s_\gamma^l\} \cup \{s_\alpha^r, s_\beta^r, s_\gamma^r\}$. To win, she will have to move Juliet towards the corresponding vertices with respect to t . Suppose she moves Romeo towards s_α^l and Juliet towards t_α^l , i.e., Romeo along $P[s, s_\alpha^l, 2M^2 + 1]$ and Juliet along $P[t, t_\alpha^l, 2M^2 + 1]$. She can move Romeo at s_α^l and Juliet at t_α^l in $2M^2 + 2$ steps. At this point, she can make one of the n choices and decide to move both Romeo and Juliet towards x_i^l for some $i \in [n]$, i.e., Romeo along $P[s_\alpha^l, x_i^l, 2M^2]$ and Juliet along $P[t_\alpha^l, x_i^l, 2M^2]$ for some $i \in [n]$. See Figure 5 for relevant vertices.

From the Divider's perspective, he can see the first choice made by Facilitator. However, he has no information about her second choice until next $2M + 2$ steps, i.e., until she moves Romeo at s_α^l and Juliet at t_α^l . Note that Facilitator can move Romeo from s_α^l to x_i^l and Juliet from t_α^l to x_i^l in $2M^2 + 1$ steps. Divider can move an agent from α_i^l to x_i^l in $2M^2$ steps. Considering the initial positions of agents, he needs to ensure that one of its agents is present on α_i^l for every $i \in [n]$ in $2M^2 + 2$ steps. For every $i \in [n]$, he needs to place an agent at u_i^j for some $j \in [m]$ and $i' \in [n]$ that he can move it to α_i^l in $2M^2 + 2$ steps. We remark that i may not be equal to i' .

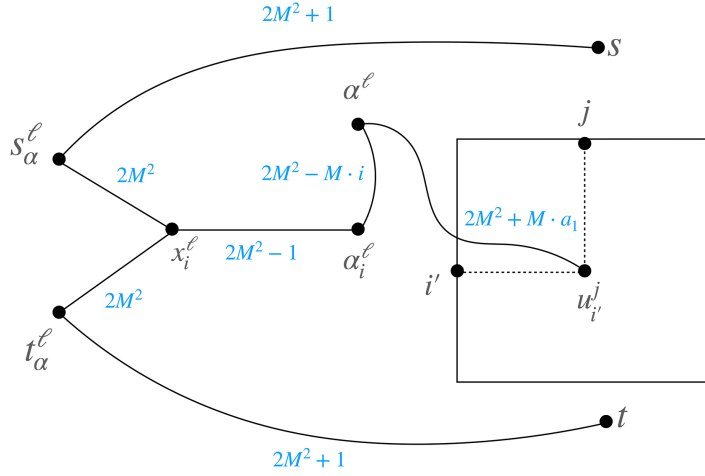


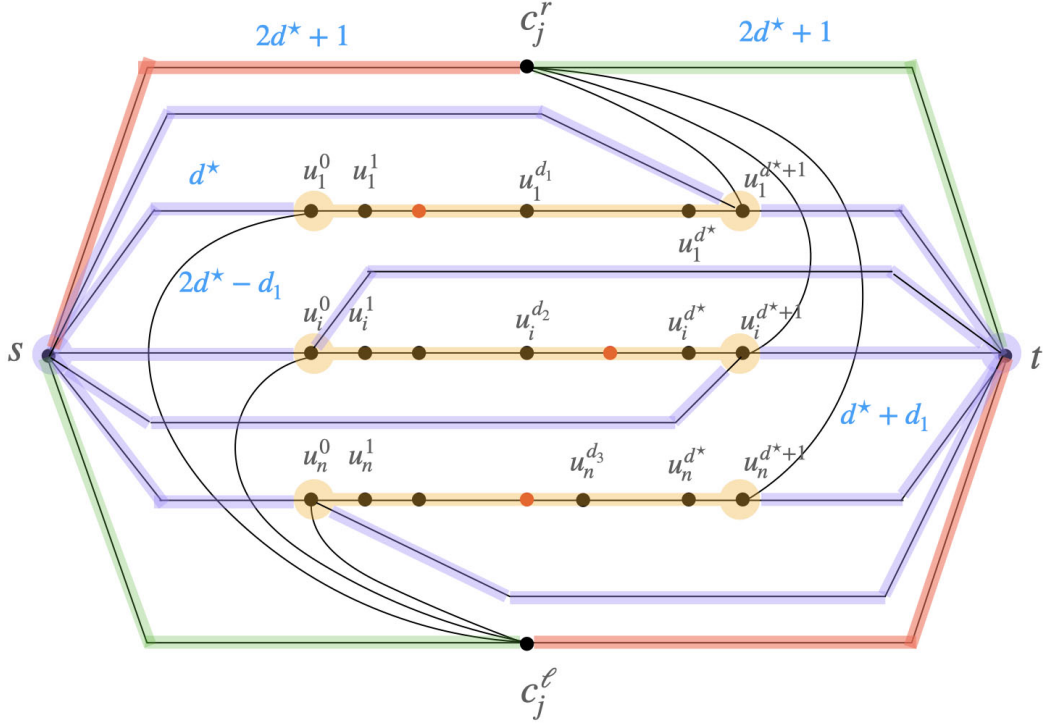
Figure 5 Vertices mentioned while presenting the intuition. The vertices near the paths indicate the number of internal vertices. Set A_j contains (α, a_1) .

The only feasible way to do so is by moving the agent from $u_{i'}^j$ to α^l and then move it from α^l to α_i^l . Suppose $(\alpha, a_1) \in A_j$ for some $a_1 \in [n]$. Recall that the number of internal vertices of path from $u_{i'}^j$ to α^l is $M^2 + M \cdot a_1$ where as that of the path from α^l to α_i^l is $M^2 - M \cdot i$. Formally, the number of internal vertices in the path $P[u_{i'}^j, \alpha^l, M^2 + M \cdot a_1] \circ P[\alpha^l, \alpha_i^l, M^2 - M \cdot i]$ is $(M^2 + M \cdot a_1) + 1 + (M^2 - M \cdot i) = 2M^2 + 1 + M^2 \cdot (a_1 - i)$. This implies that Divider can move an agent from $u_{i'}^j$ to α_i^l in $2M^2 + 2 + M^2 \cdot (a_1 - i)$ steps. Hence, for every $i \in [n]$, Divider should place an agent at $u_{i'}^j$ for some $j \in [m]$ and $i' \in [n]$ such that for $(\alpha, a_1) \in A_j$, we have $a_1 \leq i$. Using identical arguments and considering the number of internal vertices on the right side, we prove that for every $i \in [n]$, he needs to place an agent at $u_{i''}^{j'}$ for some $j' \in [m]$ and $i'' \in [n]$ such that for $(\alpha, a_2) \in A_{j'}$, we have $a_2 \geq i$. Combining these two arguments, Divider needs to place an agent at $u_{i'}^j$ such that for $(\alpha, a_1) \in A_j$, we have $a_1 = i$. Moving the agent from this position will prevent Facilitator from moving both Romeo and Juliet at x_i^l and x_i^r . This corresponds to selecting a set from \mathcal{F} to covers element $(\alpha, i) \in \mathcal{U}$. This concludes the intuition for the correctness of the reduction.

Consider set $S := \{s, t\} \cup \{s_\alpha^l, s_\beta^l, s_\gamma^l\} \cup \{\alpha^l, \beta^l, \gamma^l\} \cup \{\alpha^r, \beta^r, \gamma^r\}$ in G . It is easy to verify that $G - S$ is a forest, i.e., the feedback vertex set number of G is at most 14. Moreover, every connected component of $G - S$ is either a path or a subdivided caterpillar. The paths correspond to the paths added while encoding elements in \mathcal{U} or while adding the critical paths. The subdivided caterpillars correspond to the base gadgets, and the path added while encoding sets in \mathcal{F}' . Note that the spine of the caterpillar is the path $P[u_i^0, u_i^{m+1}, m]$ for some $i \in [n]$ added as a part of base gadget. This implies that the pathwidth of the resulting graph is at most 16.

3 co-W[1]-hardness Parameterized by FVS, Pathwidth, and the Solution Size

In this section, we prove Theorem 2 that states RENDEZVOUS is co-W[1]-hard when parameterized by the feedback vertex set number or pathwidth and the solution size. To do that, we present a parameter preserving reduction from the (MONOTONE) NAE-INTEGGER-3-SAT



■ **Figure 6** The reduction adds a yellow shaded path for each variable. Each yellow, purple, or blue shaded path has d^* many internal vertices. The green and red shaded paths have $2d^* + 1$ many internal vertices. The number of internal vertices in the remaining path in the figure depends on constants in the clause they are encoding. Note that vertices g_1, g_2 and paths $P[s, u_n^{d^*+1}, d^*], P[t, u_1^0, d^*]$ are not shown in the figure for clarity. The red vertices denote the positions of the agents.

problem. For notational convenience, we work with the following definition of the problem. An input consists of variables $\mathcal{X} = \{x_1, \dots, x_n\}$ that each take a value in the domain $\mathcal{D} = \{1, \dots, d^*\}$ and clauses $\mathcal{C} = \{C_1, \dots, C_m\}$ of the form

$$\text{NAE}(x_{i_1} \leq d_1, x_{i_2} \leq d_2, x_{i_3} \leq d_3),$$

where $d_1, d_2, d_3 \in [d^*]$. Such a clause is satisfied if not all three inequalities are **True** and not all are **False** (i.e., they are “not all equal”). The goal is to find an assignment of the variables that satisfies all given clauses. Bringmann et. al. [1] proved that (MONOTONE) NAE-INTEGGER-3-SAT is $W[1]$ -hard when parameterized by the number of variables.

Reduction

The reduction takes as input an instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ of (MONOTONE) NAE-INTEGGER-3-SAT and returns an instance (G, s, t, k) of RENDEZVOUS. We construct the graph G as follows: (See Figure 6 for the overview of the constructed graph.)

The Variable Gadget. Recall that we use $P[u, v, d]$ to denote a simple path from u to v that contains d many internal vertices. For every $i \in [n]$, it adds a simple path $P[u_i^0, u_i^{d^*+1}, d^*]$. Suppose the internal vertices of $P[u_i^0, u_i^{d^*+1}, d^*]$ are denoted by u_i^d for every $d \in [d^*]$, and u_i^0 is adjacent with u_i^1 and $u_i^{d^*+1}$ is adjacent with $u_i^{d^*}$.

The Clause Gadget. For every $j \in [m]$, the reduction adds two vertices c_j^ℓ and c_j^r . Suppose $C_j = \text{NAE}(x_{i_1} \leq d_1, x_{i_2} \leq d_2, x_{i_3} \leq d_3)$ for some $j \in [m]$. To encode the inequality $x_{i_1} \leq d_1$, the reduction adds simple paths $P[c_j^\ell, u_{i_1}^0, 2d^* - d_1]$ and $P[c_j^r, u_{i_1}^{d^*+1}, d^* + d_1]$. It encodes the other two inequalities similarly. We highlight that the number of internal vertices in these simple paths depends on the constant in the inequalities they encode.

Critical vertices and connecting paths. The reduction adds special vertices s and t and two more vertices g_1 and g_2 , and makes them common neighbours of s and t .

- For every $i \in [n]$, it adds the following simple paths:
 - $P[s, u_i^0, d^*], P[s, u_i^{d^*+1}, d^*],$
 - $P[t, u_i^0, d^*], P[t, u_i^{d^*+1}, d^*].$
- For every $j \in [m]$, it adds the following simple paths:
 - $P[s, c_j^\ell, 2d^* + 1], P[s, c_j^r, 2d^* + 1],$
 - $P[t, c_j^\ell, 2d^* + 1], P[t, c_j^r, 2d^* + 1].$

This completes the construction of the graph G . The reduction sets $k = n + 2$ and returns (G, s, t, k) as the reduced instance of RENDEZVOUS.

Intuition for the correctness

We present an intuition for the correctness of the reduction. Recall that we use $P[u, v, d_1] \circ P[v, w, d_2]$ to denote the unique path from u to w that contains v . Consider the paths $P[s, u_i^0, d^*] \circ P[u_i^0, t, d^*]$ and $P[s, u_i^{d^*+1}, d^*] \circ P[u_i^{d^*+1}, t, d^*]$ for every $i \in [n]$ and paths $P[s, c_j^\ell, 2d^* + 1] \circ P[c_j^\ell, t, 2d^* + 1]$ and $P[s, c_j^r, 2d^* + 1] \circ P[c_j^r, t, 2d^* + 1]$ for every $j \in [m]$. As we will see, the only way Facilitator can win in Rendezvous Games with Adversaries is by moving Romeo and Juliet along with one of these $2n + 2m$ paths. As g_1, g_2 are common neighbors of s and t , Divider needs to put two of the $k = n + 2$ agents on g_1 and g_2 . Suppose he puts the remaining n agents at some internal vertices of the paths added while encoding variables. He places the agents such that each path contains one of them. For example, the red vertices in Figure 6 corresponds to the positions of the agents on the paths added while encoding variables x_1, x_2 , and x_3 .

Suppose Divider places an agent at an internal vertex, say u_1^d , of $P[u_1^0, u_1^{d^*+1}, d^*]$. Facilitator can move Romeo and Juliet to either u_1^0 or $u_1^{d^*+1}$ in $d^* + 1$ steps. The length of the path from u_1^0 to u_1^d is d and the length of the path from $u_1^{d^*+1}$ to u_1^d is $d^* - d + 1$.

- Divider can move the agent from u_1^d to u_1^0 in at most d^* steps as $d \leq d^*$, and
- Divider can move the agent from u_1^d to $u_1^{d^*+1}$ in at most d^* steps as $d^* - d + 1 \leq d^*$.

Recall that the simple path $P[c_j^\ell, u_1^0, 2d^* - d_1]$, as the notation suggests, has $2d^* - d_1$ internal vertices. Hence, the path $P[c_j^\ell, u_1^0, 2d^* - d_1] \circ P[u_1^0, u_1^d, d - 1]$ has $(2d^* - d_1) + 1 + (d - 1)$ many internal vertices. Hence, the length of path from c_j^ℓ to u_1^d is $2d^* + 1 + d - d_1$.

- Divider can move the agent from u_1^d to c_j^ℓ in at most $2d^* + 1$ steps only if $d \leq d_1$.

Consider symmetric arguments for c_j^r . The simple path $P[c_j^r, u_1^{d^*+1}, d^* + d_1]$ has $d^* + d_1$ many internal vertices. Hence, the path $P[c_j^r, u_1^{d^*+1}, d^* + d_1] \circ P[u_1^{d^*+1}, u_1^d, d^* - d]$ has $(d^* + d_1) + 1 + (d^* - d)$ many internal vertices. Hence, the length of the path from u_1^d to c_j^r is $2d^* + 2 + d_1 - d$.

- Divider can move the agent from u_1^d to c_j^r in at most $2d^* + 1$ steps only if $d > d_1$.

Suppose there is a clause $C_j \in \mathcal{C}$ such that $C_j = \text{NAE}(x_1 \leq d_1, x_2 \leq d_2, x_3 \leq d_3)$. Consider the two vertices c_j^ℓ and c_j^r added while encoding C_j . Note that Facilitator can move Romeo and Juliet to either c_j^ℓ or c_j^r in $2d^* + 2$ steps. Moreover, apart from s and t , the only

branching points in paths $P[s, c_j^\ell, 2d^* + 1] \circ P[c_j^\ell, t, 2d^* + 1]$ and $P[s, c_j^r, 2d^* + 1] \circ P[c_j^r, t, 2d^* + 1]$ are c_j^ℓ and c_j^r , respectively. Hence, Divider needs to place an agent that he can move to c_j^ℓ in at most $2d^* + 1$ steps. Similarly, he needs to place an agent that he can move to c_j^r in at most $2d^* + 1$ steps. As we will see, Divider can only move the agents stationed at the paths corresponding to variables x_1, x_2 , or x_3 to c_j^ℓ or c_j^r in at most $2d^* + 1$ steps. Hence, he needs to place agents at the interior vertices, say $u_1^{c_1}, u_2^{c_2}, u_3^{c_3}$, of $P[u_1^0, u_1^{d^*+1}, d^*]$, $P[u_2^0, u_2^{d^*+1}, d^*]$ and $P[u_3^0, u_3^{d^*+1}, d^*]$, respectively, such that

- at least one of the inequalities in $\{c_1 \leq d_1; c_2 \leq d_2; c_3 \leq d_3\}$ is **True**, and
- *simultaneously* at least one of the inequalities in $\{c_1 > d_1; c_2 > d_2; c_3 > d_3\}$ is **True**.

This position of agents corresponds to the value of variables x_1, x_2, x_3 in $[d^*]$ that satisfy the clause $C_j = \text{NAE}(x_1 \leq d_1, x_2 \leq d_2, x_3 \leq d_3)$. In the following two lemmas, we formalize these intuitions.

► **Lemma 5.** *If $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a YES-instance of (MONOTONE) NAE-INTEGGER-3-SAT, then $(G, s, t, n + 2)$ is a NO-instance of RENDEZVOUS.*

Proof. We show that if $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a YES-instance of (MONOTONE) NAE-INTEGGER-3-SAT, then Divider with $n + 2$ agents can win in Rendezvous Game with Adversaries. Recall that $n = |\mathcal{X}|$, and $m = |\mathcal{C}|$. Suppose $\psi : \mathcal{X} \rightarrow [d^*]$ be a satisfying assignment, and $\psi(x_i) = d_i$ for every $i \in [n]$.

We describe a winning strategy for Dividers with the agents D_1, D_2, \dots, D_{n+2} . Initially, he puts D_i in the vertex $u_i^{d_i}$, for every $i \in [n]$, and D_{n+1} and D_{n+2} in g_1 and g_2 respectively. He does not move agents D_1, \dots, D_{n+2} , until Facilitator moves Romeo or Juliet from s or t , respectively. Suppose without loss of generality Facilitator first moves Romeo from s (she may or may not move Juliet from t). By the construction, she can move Romeo either on the paths $P[s, u_i^0, d^*]$, $P[s, u_i^{d^*+1}, d^*]$ for some $i \in [n]$ or on the paths $P[s, c_j^\ell, 2d^* + 1]$, $P[s, c_j^r, 2d^* + 1]$ for some $j \in [m]$.

Suppose Facilitator moves Romeo from s to a vertex on the path $P[s, u_i^0, d^*]$ for some $i \in [n]$. Divider moves D_{n+1} from g_1 to s and then towards u_i^0 as she moves Romeo towards u_i^0 . He also moves D_i to u_i^0 in at most $\psi(x_i)$ steps along the path $P[u_i^0, u_i^{d^*+1}, d^*]$. Facilitator needs at least $d^* + 1$ steps to move both Romeo and Juliet in u_i^0 starting from s and t respectively. As $\psi(x_i) \leq d^*$, Divider can move D_i to u_i^0 before Facilitator can move both Romeo and Juliet to u_i^0 . Hence, he can block Romeo by D_i and D_{n+1} on the path $P[s, u_i^0, d^*]$. Divider keeps moving D_i and D_{n+1} towards Romeo's position and in at most $\psi(x_i) + d^* - 1$ steps Facilitator can not move Romeo. This implies Divider wins by keeping Romeo in its current position with its neighbors occupied by D_i and D_{n+1} . The argument also follows when Facilitator moves Romeo from s to a vertex on the path $P[s, u_i^{d^*+1}, d^*]$ for some $i \in [n]$ since Divider can move D_i to $u_i^{d^*+1}$ in at most $d^* - \psi(x_i) + 1 (\leq d^*)$ steps.

Suppose Facilitator moves Romeo from s to a vertex on the path $P[s, c_j^\ell, 2d^* + 1]$ for some $j \in [m]$. Let $C_j = \text{NAE}(x_{i_1} \leq d_1, x_{i_2} \leq d_2, x_{i_3} \leq d_3)$. Since ψ is a satisfying assignment, it sets the values of variables such that at least one of the inequalities will be **True** and at least one of the inequalities will be **False**. We assume without loss of generality that $\psi(x_{i_1}) \leq d_1$ and $\psi(x_{i_2}) > d_2$. Divider moves D_{i_1} to c_j^ℓ in at most $2d^* - d_1 + 1 + \psi(x_{i_1})$ steps through the path $P[c_j^\ell, u_{i_1}^0, 2d^* - d_1] \circ P[u_{i_1}^0, u_{i_1}^{d^*+1}, d^*]$. As in the previous case, he can move D_{n+1} from g_1 to s and then keep moving towards c_j^ℓ as Facilitator moves Romeo towards c_j^ℓ . He can move D_{n+2} in a similar manner with respect to Juliet.

Facilitator can move both Romeo and Juliet to c_j^ℓ in at least $2d^* + 2$ steps starting from s and t respectively. Divider can move D_{i_1} to c_j^ℓ before Romeo and Juliet as $2d^* - d_1 + 1 + \psi(x_{i_1}) \leq 2d^* + 1$. Hence, Romeo is blocked by D_{i_1} and D_{n+1} on the path $P[s, c_j^\ell, 2d^* + 1]$

and Juliet cannot reach Romeo. Divider keeps moving D_{i_1} and D_{n+1} towards Romeo and in at most $4d^* - d_1 + 1 + \psi(x_{i_1})$ steps Romeo cannot move. This implies Divider wins. The argument also follows when Facilitator moves Romeo from s to a vertex on the path $P[s, c_j^r, 2d^* + 1]$ for some $j \in [m]$ since Divider can move D_{i_2} to c_j^r in at most $2d^* + 2 + d_2 - \psi(x_{i_2})$ ($< 2d^* + 2$) steps. This implies that if $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a YES-instance of (MONOTONE) NAE-INTEGER-3-SAT, then Divider with $n + 2$ agents can win in Rendezvous Game with Adversaries, i.e., $(G, s, t, n + 2)$ is a NO-instance of RENDEZVOUS. ◀

► **Lemma 6.** *If $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a NO-instance of (MONOTONE) NAE-INTEGER-3-SAT, then $(G, s, t, n + 2)$ is a YES-instance of RENDEZVOUS.*

Proof. We show that if $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a NO-instance of (MONOTONE) NAE-INTEGER-3-SAT, then Facilitator wins in at most $2d^* + 2$ steps against Divider with $n + 2$ agents.

We first consider two simple cases where Facilitator has an easy winning strategy. First, consider the case when Divider does not place his agents at g_1 or g_2 . Then, she can move Romeo and Juliet there and win in one step. Second, consider the case when there is $i \in [n]$ such that none of Divider's agents is within distance d^* from u_i^0 or from $u_i^{d^*+1}$. In the first sub-case, she can move Romeo and Juliet to u_i^0 in $d^* + 1$ steps through the paths $P[s, u_i^0, d^*]$ and $P[t, u_i^0, d^*]$, respectively, and win. Similarly, in the second sub-case she can move Romeo and Juliet to $u_i^{d^*+1}$ in $d^* + 1$ steps through the paths $P[s, u_i^{d^*+1}, d^*]$ and $P[t, u_i^{d^*+1}, d^*]$, respectively, and win.

In the remaining proof, we suppose that Divider places D_{n+1} at g_1 and D_{n+2} at g_2 . Moreover, for every $i \in [n]$, there is a Divider's agent within distance d^* from u_i^0 and within distance d^* from $u_i^{d^*+1}$. Suppose from now that for every $i \in [n]$, there exists a Divider's agent within distance d^* from u_i^0 and within distance d^* from $u_i^{d^*+1}$. By the construction and the fact that Divider can not place an agent at s or t , a single Divider's agent cannot be within distance d^* from both u_i^0 and u_j^0 , or $u_i^{d^*+1}$ and $u_j^{d^*+1}$, or u_i^0 and $u_j^{d^*+1}$, for $i \neq j \in [n]$. As Divider has n remaining agents, for every $i \in [n]$, there must be an agent, say D_i , within distance d^* from both u_i^0 and $u_i^{d^*+1}$. This is possible only when for every $i \in [n]$, D_i is on one of the internal vertices of the path $P[u_i^0, u_i^{d^*+1}, d^*]$. Suppose $\phi : [n] \rightarrow [d^*]$ is the mapping corresponding to the initial position of the Divider's agents. Formally, for every $i \in [n]$, Divider places agent D_i on $u_i^{\phi(i)}$. For every $i \in [n]$, the initial position of D_i also represents a possible assignment of variable x_i in $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

We now define the Facilitator's strategy. Considering $\mathcal{X} = \{x_1, \dots, x_n\}$ as the variables that each take a value in the domain $\mathcal{D} = \{1, \dots, d^*\}$, she constructs a collection \mathcal{C} of clauses such that for every $j \in [m]$, clause $C_j = \text{NAE}(x_{i_1} \leq d_1, x_{i_2} \leq d_2, x_{i_3} \leq d_3)$, where $x_{i_1}, x_{i_2}, x_{i_3} \in \mathcal{X}$ for some $d_1, d_2, d_3 \in [d^*]$. Alternately, she reverse-engineers the process used by the reductions to encode clauses. She also constructs an assignment $\psi : \mathcal{X} \rightarrow \mathcal{D} = [d^*]$ by considering the initial positions of agents D_1, D_2, \dots, D_n . Formally, $\psi(x_i) = \phi(i)$ for every $i \in [n]$. It then determines whether the following statements are **True**.

1. For some clause $C_j = \text{NAE}(x_{i_1} \leq d_1, x_{i_2} \leq d_2, x_{i_3} \leq d_3)$, all of the inequalities in $\{\psi(x_{i_1}) \leq d_1; \psi(x_{i_2}) \leq d_2; \psi(x_{i_3}) \leq d_3\}$ are **True**, where $j \in [m]$.
2. For some clause $C_j = \text{NAE}(x_{i_1} \leq d_1, x_{i_2} \leq d_2, x_{i_3} \leq d_3)$, all of the inequalities in $\{\psi(x_{i_1}) \leq d_1; \psi(x_{i_2}) \leq d_2; \psi(x_{i_3}) \leq d_3\}$ are **False**, where $j \in [m]$.

Facilitator has to make a critical choice in the first step where she has to decide about moving Romeo towards $c_1^l, \dots, c_m^l, c_1^r, \dots, c_m^r$. This choice depends on which of the above statement is **True** and for which clause it is **True**. If the first statement is **True** for the clause $C_j \in \mathcal{C}$, then she moves Romeo and Juliet towards c_j^r . Similarly, if the second statement is **True** for the clause $C_j \in \mathcal{C}$, then she moves Romeo and Juliet towards c_j^l .

To argue that this is indeed a winning strategy for Facilitator, we first argue that for any initial positions of Divider's agents, at least one of the two statements above is **True**. Assume the above two statements are **False** for all $j \in [m]$, which implies in all the clauses $C_j \in \mathcal{C}$, not all three inequalities are **True** and not all are **False**. Hence, all the clauses are satisfied by the assignment ψ . This, however, contradicts the fact that $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a NO-instance. Hence, for any initial positions of Divider's agents, at least one of the two sentences is **True**.

This allows Facilitator to make her choice. It remains to argue that Romeo and Juliet can meet at the vertex c_j^ℓ or c_j^r which Facilitator has chosen. Suppose, one of the statements is **True** for the clause C_j . For notational convenience, suppose $C_j = \text{NAE}(x_1 \leq d_1, x_2 \leq d_2, x_3 \leq d_3)$.

Suppose $\psi(x_1) \leq d_1, \psi(x_2) \leq d_2, \psi(x_3) \leq d_3$ (i.e. First statement is **True**). Then, as mentioned in the Facilitator's strategy, her choice will be to move Romeo and Juliet towards c_j^r . For $i \in \{1, 2, 3\}$, Divider needs at least $d^* - \psi(x_i) + 1 + d^* + d_i + 1 \geq 2d^* + 2$ steps to move D_i from $u_i^{\psi(x_i)}$ to c_j^r via the shortest path $P[u_i^{\psi(x_i)}, u_i^{d^*+1}, d^* - \psi(x_i)] \circ P[u_i^{d^*+1}, c_j^r, d^* + d_i]$. Note that, by the construction, the Divider's agents that are at distance less than or equal to $2d^* + 2$ from c_j^r are D_1, D_2 and D_3 , only. Facilitator can move Romeo and Juliet to c_j^r in $2d^* + 2$ steps through the paths $P[s, c_j^r, 2d^* + 1]$ and $P[t, c_j^r, 2d^* + 1]$, respectively. Since Facilitator takes the first turn, she can move Romeo and Juliet to c_j^r before Divider's agents. Hence, Facilitator wins in $2d^* + 2$ steps.

Suppose $\psi(x_1) > d_1, \psi(x_2) > d_2, \psi(x_3) > d_3$ (i.e. Second statement is **True**). Then, as mentioned in the Facilitator's strategy, her choice will be to move Romeo and Juliet towards c_j^ℓ . For $i \in \{1, 2, 3\}$, Divider needs at least $\psi(x_i) - 1 + 1 + 2d^* - d_i + 1 > 2d^* + 1$ steps to move D_i from $u_i^{\psi(x_i)}$ to c_j^ℓ via the shortest path $P[u_i^{\psi(x_i)}, u_i^0, \psi(x_i) - 1] \circ P[u_i^0, c_j^\ell, 2d^* - d_i]$. Once again, by the construction, the Divider's agents that are at distance less than or equal to $2d^* + 2$ from c_j^ℓ are D_1, D_2 and D_3 . Facilitator moves Romeo and Juliet to c_j^ℓ in $2d^* + 2$ steps through the paths $P[s, c_j^\ell, 2d^* + 1]$ and $P[t, c_j^\ell, 2d^* + 1]$ respectively. Since Facilitator takes the first turn, Romeo and Juliet is moved to c_j^ℓ before Divider agents and Facilitator wins in $2d^* + 2$ steps.

This implies that if $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a NO-instance of (MONOTONE) NAE-INTEGGER-3-SAT, then Facilitator wins in at most $2d^* + 2$ steps against Divider with $n + 2$ agents, i.e., $(G, s, t, n + 2)$ is a YES-instance of RENDEZVOUS. ◀

By the construction, the number of agents is upper bounded by the number of variables in (MONOTONE) NAE-INTEGGER-3-SAT plus two. Consider the set $S := \bigcup_{i \in [n]} \{u_i^0, u_i^{d^*+1}\} \cup \{s, t\}$ of $2n + 2$ vertices in G . It is easy to verify that $G - S$ is a collection of paths (corresponding to variable gadgets) and subdivided stars (centered at the vertices added while encoding the clauses). It is easy to verify that the pathwidth of a subdivided star is at most two. Hence, the feedback vertex set number and the pathwidth of the resulting graph are bounded by the linear function in the number of variables. Lemma 5, Lemma 6 and the fact that the reduction can be completed in the polynomial time in the size of input imply Theorem 2 which we restate here.

- **Theorem 2.** RENDEZVOUS is co-W[1]-hard when parameterized by:
- the feedback vertex set number and the solution size, or
 - the pathwidth and the solution size.

4 FPT Parameterized by the Vertex Cover Number and Solution Size

In this section we focus on Theorem 3. Throughout this section, we assume that a vertex cover X of size $\text{vc}(G)$ is given as a part of the input. We describe the FPT result here and defer the non-existence of the polynomial kernel, which follows from observing the properties of a known reduction in [4], to the full version [6].

► **Theorem 3.** RENDEZVOUS is FPT when parameterized by the vertex cover number of the input graph and the solution size. Moreover, the problem does not admit a polynomial kernel when parameterized by the vertex cover number and the solution size unless $\text{NP} \subseteq \text{co-NP}/\text{poly}$.

► **Reduction Rule 4.1.** Consider an instance (G, X, s, t, k) of RENDEZVOUS. If $s = t$, $st \in E(G)$, $|N(s) \cap N(t)| > k$, then return a trivial YES-instance.

For the rest of this discussion, we will assume that any instance (G, s, t, k) of RENDEZVOUS under consideration does *not* satisfy the premise of Reduction Rule 4.1, i.e, we assume that we are not dealing with trivial YES instances. Also, since the vertices s and t can always be added to the vertex cover and this only increases the parameter by two, we assume for simplicity – and without loss of generality – that $s, t \in X$.

We now introduce some notation. For a subset $Y \subseteq X$, let $I_Y \subseteq G \setminus X$ denote the set of vertices in $G \setminus X$ whose neighborhood is exactly Y . Note that $\{I_Y\}_{Y \subseteq X}$ is a partition of $G \setminus X$ into at most $2^{\text{vc}(G)}$ many parts. For a vertex $v \in G \setminus X$, we use $\mathcal{E}_{G,X}(v)$ to denote the part that v belongs to, in other words, $\mathcal{E}_{G,X}(v) = I_{N(v)}$. We now apply the following reduction rule.

► **Reduction Rule 4.2.** Consider an instance (G, X, s, t, k) of RENDEZVOUS. Repeat the following for each $v \in G \setminus X$. If $|\mathcal{E}_{G,X}(v)| > k + 1$, then choose any subset of exactly $k + 1$ vertices from $\mathcal{E}_{G,X}(v)$ and delete rest of the vertices from $\mathcal{E}_{G,X}(v)$.

► **Lemma 7.** Reduction Rule 4.2 is safe.

Proof. Let (G, X, s, t, k) denote the input instance, and let $v \in G \setminus X$ be arbitrary but fixed. Further, let (H, X, s, t, k) denote the instance obtained by applying Reduction Rule 4.2 with respect to v . If $|\mathcal{E}(v)| \leq k + 1$ in G then $G = H$ and there is nothing to prove. Otherwise, let $Q_v \subseteq \mathcal{E}_{G,X}(v)$ denote the set of vertices deleted by the application of the reduction rule with respect to v . Note that $H = G \setminus Q_v$. Also observe that $|\mathcal{E}_{H,X}(v)| = k + 1$.

To begin with, suppose the Facilitator has a winning strategy in G . Observe that the Facilitator can employ the same strategy in H as well, except when the strategy involves moving to a vertex $u \in Q_v$. However, since $|\mathcal{E}_{H,X}(v)| = k + 1$, we have that there is at least one vertex w in $H \setminus X$ that has the same neighborhood as u and is not occupied by an agent of the Divider, since the Divider has only k agents at their disposal. The strategy, at this point, would remain valid if we were to replace u with w . If the strategy involved using two distinct vertices from Q_v in the same step, then note that we can modify the strategy and have the Facilitator's agents meet immediately at the vertex w .

On the other hand, if the Facilitator had a winning strategy in H , then it is easy to check that the Facilitator can win in G by mimicing the strategy directly. Another way to see this is the following. Suppose that the Divider had a winning strategy in G . Then observe that in any step, without loss of generality, if the Divider's agents occupy some vertices of $\mathcal{E}_{G,X}(v)$, we can replace this configuration with all of these agents on a single vertex of $\mathcal{E}_{G,X}(v)$ outside Q_v . Thus any winning strategy for the divider in G can be adapted to a valid winning strategy in H . This concludes the argument for the equivalence of the two instances. ◀

► **Lemma 8.** RENDEZVOUS is FPT when parameterized by the vertex cover number and the solution size.

Proof. Observe that repeated applications of Reduction Rule 4.2 ensures that $|V(G)| = |X| + |G \setminus X| \leq \text{vc}(G) + 2^{\text{vc}(G)} \cdot (k + 1)$. Thus we have an exponential kernel in $\text{vc}(G)$, and the claim follows. ◀

5 Conclusion

In this work, we studied the game of rendezvous with adversaries on a graph introduced by Fomin, Golovach, and Thilikos [4]. The game is a natural dynamic version of the problem of finding a vertex cut between two vertices s and t . Given that the problem is $W[2]$ -hard when parameterized by the natural parameter, i.e. the solution size, we continued studying structural parameters of the input graph initiated by Fomin et al. [4]. We proved, to our surprise, that the problem is **co-NP-hard** even when restricted to graphs whose feedback vertex set number is at most 14, or pathwidth is at most 16. In particular, we proved **RENDEZVOUS** is **co-para-NP-hard** parameterized by treewidth, thereby answering an open question by Fomin et al. [4]. It turns out that even augmenting the feedback vertex set number or the pathwidth with the solution size is not enough. Specifically, we proved that **RENDEZVOUS** is **co- $W[1]$ -hard** when parameterized by the feedback vertex set number and the solution size, or the pathwidth and the solution size. Towards the positive side, we proved that the problem admits a natural exponential kernel when parameterized by the vertex cover number and the solution size, however this kernel cannot be improved to a polynomial kernel under standard complexity-theoretic assumptions. Finally, we presented polynomial time algorithms on two restricted cases and proved that **RENDEZVOUS** can be solved in polynomial time on the classes of treewidth at most two graphs and grids.

While we addressed the structural parameterized by arguably the most well studied parameters, it remains interesting to study the parameterized complexity by other structural parameters. Amongst these, we highlight the following question: *Is RENDEZVOUS $W[1]$ -hard when parameterized by the vertex cover number (only)?* We tend to believe it is indeed the case. To the best of our knowledge, the problems that are $W[1]$ -hard when parameterized by the vertex cover number, like **LIST COLORING**, **WEIGHTED (k, r) -CENTER**, etc., have additional input arguments like lists or weights. We believe that the dynamic nature of the **RENDEZVOUS** problem might make it an exception to the above known trend.

References

- 1 Karl Bringmann, Danny Hermelin, Matthias Mnich, and Erik Jan van Leeuwen. Parameterized complexity dichotomy for steiner multicut. *J. Comput. Syst. Sci.*, 82(6):1020–1043, 2016. doi:10.1016/j.jcss.2016.03.003.
- 2 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.
- 3 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 4 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Can romeo and juliet meet? or rendezvous games with adversaries on graphs. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021, Warsaw, Poland, June 23-25, 2021, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 308–320. Springer, 2021. doi:10.1007/978-3-030-86838-3_24.
- 5 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 6 Neeldhara Misra, Manas Mulpuri, Prafullkumar Tale, and Gaurav Viramgami. Romeo and juliet meeting in forest like regions, 2022. doi:10.48550/arXiv.2210.02582.

A Appendix

A.1 Preliminaries

For a positive integer q , we denote the set $\{1, 2, \dots, q\}$ by $[q]$. We use \mathbb{N} to denote the collection of all non-negative integers.

Graph theory

We use standard graph-theoretic notation, and we refer the reader to [3] for any undefined notation. For an undirected graph G , sets $V(G)$ and $E(G)$ denote its set of vertices and edges, respectively. We denote an edge with two endpoints u, v as uv . Unless otherwise specified, we use n to denote the number of vertices in the input graph G of the problem under consideration. Two vertices u, v in $V(G)$ are *adjacent* if there is an edge uv in G . The *open neighborhood* of a vertex v , denoted by $N_G(v)$, is the set of vertices adjacent to v . The *closed neighborhood* of a vertex v , denoted by $N_G[v]$, is the set $N_G(v) \cup \{v\}$. We say that a vertex u is a *pendant vertex* if $|N_G(v)| = 1$. The *degree* of a vertex v , denoted by $\deg_G(v)$, is equal to the number of vertices in the open neighbourhood of v , i.e., $\deg_G(v) = |N_G(v)|$. We omit the subscript in the notation for neighborhood if the graph under consideration is clear.

For a subset S of $V(G)$, we define $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$. For a subset F of edges, we denote by $V(F)$ the collection of endpoints of edges in F . For a subset S of $V(G)$ (resp. a subset F of $E(G)$), we denote the graph obtained by deleting S (resp. deleting F) from G by $G - S$ (resp. by $G - F$). We denote the subgraph of G induced on the set S by $G[S]$.

A graph is *connected* if there is a path between every pair of distinct vertices. A subset $S \subseteq V(G)$ is said to be a *connected set* if $G[S]$ is connected.

A *simple path*, denoted by $P[u, v, d]$, is a non-empty graph G of the form $V(G) = \{u, x_1, \dots, x_d, v\}$, and $E(G) = \{ux_1, x_1x_2, \dots, x_{d-1}x_d, x_dv\}$, where u, v , and all x_i 's are distinct. The vertices $\{x_1, x_2, \dots, x_d\}$ are the *internal vertices* of $P[u, v, d]$, and the vertices $\{x_i : \deg(x_i) > 2, i \in [d]\}$, i.e., internal vertices whose *degree* is strictly greater than 2 are the *branching points* of $P[u, v, d]$. We use $P[u, v, d_1] \circ P[v, w, d_2]$ to denote the unique simple path from u to w that contains v and has $d_1 + d_2 + 1$ many internal vertices.

A set of vertices Y is said to be an *independent set* if no two vertices in Y are adjacent. For a graph G , a set $X \subseteq V(G)$ is said to be a *vertex cover* if $V(G) \setminus X$ is an independent set. A set of vertices Y is said to be a *clique* if any two vertices in Y are adjacent. A vertex cover X is a *minimum vertex cover* if for any other vertex cover Y of G , we have $|X| \leq |Y|$. We denote by $\text{vc}(G)$ the size of a minimum vertex cover of a graph G . For a graph G , a set $X \subseteq V(G)$ is said to be a *feedback vertex set* if $V(G) \setminus X$ does not contain a cycle. We denote by $\text{fvs}(G)$ the size of a minimum feedback vertex set of a graph G .

A *path decomposition* of a graph G is a sequence $\mathcal{P} = (X_1, X_2, \dots, X_r)$ of *bags*, where $X_i \subseteq V(G)$ for each $i \in [r]$, such that the following conditions hold:

- $\bigcup_{i=1}^r X_i = V(G)$. In other words, every vertex of G is in at least one bag.
- For every $uv \in E(G)$, there exists $\ell \in [r]$ such that the bag X_ℓ contains both u and v .
- For every $u \in V(G)$, if $u \in X_i \cap X_k$ for some $i \leq k$, then $u \in X_j$ also for each j such that $i \leq j \leq k$. In other words, the indices of the bags containing u form an interval in $[r]$.

The *width* of a path decomposition (X_1, X_2, \dots, X_r) is $\max_{1 \leq i \leq r} |X_i| - 1$. The *pathwidth* of a graph G , denoted by $\text{pw}(G)$, is the minimum possible width of a path decomposition of G .

27:18 Romeo and Juliet Meeting in Forest like Regions

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a *bag*, such that the following conditions hold:

- $\bigcup_{t \in V(T)} X_t = V(G)$. In other words, every vertex of G is in at least one bag.
- For every $uv \in E(G)$, there exists a node t of T such that bag X_t contains both u and v .
- For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$, i.e., the set of nodes whose corresponding bags contains u , induces a connected subtree of T .

The *width* of a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of a graph G , denoted by $\text{tw}(G)$, is the minimum possible width of a tree decomposition of G .

A $M \times N$ *grid* is the graph G of the form $V(G) = \{(i, j) : i \in [M], j \in [N]\}$, and $E(G) = \{(i, j)(i', j') : |i - i'| + |j - j'| = 1, i, i' \in [M], j, j' \in [N]\}$.

Let X and Y be multisets of vertices of a graph G (i.e., X and Y can contain several copies of the same vertex). We say that X and Y of the same size are *adjacent* if there is a bijective mapping $\alpha : X \rightarrow Y$ such that for $x \in X$, either $x = \alpha(x)$ or x and $\alpha(x)$ are adjacent in G .

Parameterized complexity

An instance of a parameterized problem Π consists of an input I , which is an input of the non-parameterized version of the problem, and an integer k , which is called the *parameter*. A problem Π is said to be *fixed-parameter tractable*, or **FPT**, if given an instance (I, k) of Π , we can decide whether (I, k) is a YES-instance of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$. Here, $f : \mathbb{N} \mapsto \mathbb{N}$ is some computable function depending only on k . Parameterized complexity theory provides tools to rule out the existence of **FPT** algorithms under plausible complexity-theoretic assumptions. For this, a hierarchy of parameterized complexity classes $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \cdots \subseteq \text{XP}$ was introduced, and it was conjectured that the inclusions are proper. The most common way to show that it is unlikely that a parameterized problem admits an **FPT** algorithm is to show that it is **W[1]** or **W[2]**-hard. It is possible to use reductions analogous to the polynomial-time reductions employed in classical complexity. Here, the concept of **W[1]**-hardness replaces the one of **NP**-hardness, and we need not only to construct an equivalent instance **FPT** time, but also to ensure that the size of the parameter in the new instance depends only on the size of the parameter in the original instance. These types of reductions are called *parameter preserving reductions*. For a detailed introduction to parameterized complexity and related terminologies, we refer the reader to the recent book by Cygan et al. [2].

A *reduction rule* is a polynomial-time algorithm that takes as input an instance of a problem and outputs another, usually reduced, instance. A reduction rule said to be *applicable* on an instance if the output instance and input instance are different. A reduction rule is *safe* if the input instance is a YES-instance if and only if the output instance is a YES-instance.

A *kernelization* of a parameterized problem Π_1 is a polynomial algorithm that maps each instance (I_1, k_1) of Π_1 to an instance I of Π_2 such that (1) (I, k) is a YES-instance of Π_1 if and only if I_2 is a YES-instance of Π_2 , and (2) the size of I_2 is bounded by $g(k)$ for a computable function $g(\cdot)$. We say a compression is a *polynomial compression* if $g(\cdot)$ is a polynomial function, then we call it a *polynomial kernel*. It is known that a problem is **FPT** if and only if it admits a kernel (See, for example, [2, Lemma 2.2]).

B Some Polynomial Cases

In this section, we focus on Theorem 4.

► **Theorem 4.** RENDEZVOUS can be solved in polynomial time on the classes of treewidth at most two graphs and grids.

We focus on graphs of treewidth at most two. In this case, we show that $d_G(s, t) = \lambda_G(s, t)$, which leads to RENDEZVOUS being polynomially solvable on this class of graphs based on standard algorithms for computing $\lambda_G(s, t)$.

► **Proposition 9.** If G is a connected graph of tree-width at most 2, then for every $s, t \in V(G)$, $d_G(s, t) = \lambda_G(s, t)$.

Proof. Consider an instance (G, s, t, k) of RENDEZVOUS where G is a graph of treewidth at most 2. We recall that these are the series-parallel graphs, which are graphs with two distinguished vertices called terminals, formed recursively by two simple composition operations. Specifically, we have the following definitions. A two-terminal graph (TTG) is a graph with two distinguished vertices, s and t called source and sink, respectively. The parallel composition $P_c = P_c(X, Y)$ of two TTGs X and Y is a TTG created from the disjoint union of graphs X and Y by merging the sources of X and Y to create the source of P_c and merging the sinks of X and Y to create the sink of P_c . The series composition $S_c = S_c(X, Y)$ of two TTGs X and Y is a TTG created from the disjoint union of graphs X and Y by merging the sink of X with the source of Y . The source of X becomes the source of S_c and the sink of Y becomes the sink of S_c . A two-terminal series-parallel graph (TTSPG) is a graph that may be constructed by a sequence of series and parallel compositions starting from a set of copies of a single-edge graph K_2 with assigned terminals. Finally, a graph is called series-parallel (SP-graph), if it is a TTSPG when some two of its vertices are regarded as source and sink.

The proof will proceed by induction on the number of vertices. We will do a case analysis for sequence of compositions used to arrive at the final graph G . In the base case, there is nothing to prove since G is simply an edge. We use x and y to denote the source and sink terminals, respectively. For the induction hypothesis, we assume that the claim is true for all series-parallel graphs with less than k vertices, where $k \geq 2$. Now, let G be a series-parallel graph having k vertices.

Suppose G is obtained by a series or parallel composition of graphs G_1 and G_2 and let x and y denote the source and sink terminals of G , while x_b and y_b denote the source and sink terminals of G_b for $b \in \{1, 2\}$. Note that G_1 and G_2 are series-parallel graphs having less than k vertices.

Case 1: $s \in G_1$ and $t \in G_2$; $s \neq x_1, s \neq y_1; t \neq x_2, t \neq y_2$

Case 1A: The composition is series. In this case static and dynamic separation number is one: $\{x\}$ or $\{y\}$ (the joined terminal).

Case 1B: The composition is parallel. Consider the path $s \rightarrow x \rightarrow t \rightarrow y \rightarrow s$. Since $s \neq t, s \neq x, s \neq y, t \neq x, t \neq y$, the considered path is a closed walk containing s and t which forms a cycle. So, s and t lies on a cycle. So, the lower bound on the dynamic separator is 2. And the upper bound on the dynamic separator is also 2 as the static separator in this case is 2. So in this case static and dynamic separation number is two and is given by both terminals together: $\{x, y\}$.

Case 2: $s \in G_1$ and $t \in G_1$ $s \neq x_1, s \neq y_1; t \neq x_1, t \neq y_1$

Case 2A: The composition is series. In this case, suppose y_1 and x_2 are identified as $g_{1,2}$; and $x = x_1$ and $y = y_2$.

27:20 Romeo and Juliet Meeting in Forest like Regions

▷ Claim 10. Static s, t separators in G_1 will also work in G and vice versa.

Proof. Forward Direction. Suppose G_1 has a static (s, t) separator S_1 of size k_1 . So, there does not exist any path from s to t in G_1 which does not contain any vertex of S_1 . Now, for the supergraph G , all the paths between s and t that does not pass through G_2 are already blocked by the static separator S_1 . Further, the paths that pass through G_2 will pass through the terminal vertex twice. So these paths will be $s \rightarrow g_{1,2} \rightarrow$ some vertices of $G_2 \rightarrow g_{1,2} \rightarrow t$. Suppose these paths are not blocked by S_1 , then there also exist a path $s \rightarrow g_{1,2} \rightarrow t$ in G_1 that are not blocked by S_1 , which contradicts the assumption that S_1 is a static (s, t) separator in G_1 . So, these paths are also blocked by S_1 , which implies that S_1 is also the static separator of G .

Backward Direction. Suppose G has a static (s, t) separator S_1 of size k_1 . Taking the vertices from G_2 in the static separator can only block paths of the type $s \rightarrow g_{1,2} \rightarrow$ some vertices of $G_2 \rightarrow g_{1,2} \rightarrow t$. So, S_1 can not contains more than one vertex from the graph G_2 , else those vertices can be replaced by $g_{1,2}$ which will result in a smaller sized static (s, t) separator of G . Hence, S_1 can contain at max one vertex from G_2 . Observe that if S_1 does not contain any vertex of G_2 , then the same S_1 is also a static (s, t) separator in G_1 . If S_1 contains exactly one vertex from G_2 , then that vertex can be replaced by $g_{1,2}$ to form a same sized static (s, t) separator in G_1 . ◁

▷ Claim 11. $d_G(s, t) = \lambda_G(s, t) = k_1$.

Proof. Suppose the claim is not true. Then we need fewer than k_1 guards in G , say $k_1 - 1$ guards are enough to separate s from t in G . But then this will also be a valid strategy in G_1 , contradicting the induction hypothesis from which we know that $d_{G_1}(s, t) = \lambda_{G_1}(s, t) = k_1$. ◁

Case 2B: The composition is parallel. In this case, we have that y_1 and y_2 join into y and x_1 and x_2 join into x .

▷ Claim 12. $\lambda_{G_1}(s, t) \leq \lambda_G(s, t) \leq \lambda_{G_1}(s, t) + 1$.

Proof. The first inequality follows from the fact that G is a supergraph of G_1 . For the second inequality, note that a separation of s and t in G can be achieved by adding one of the terminal vertices to the static separator in G_1 . ◁

▷ Claim 13. $k_1 \leq d_G(s, t) = \lambda_G(s, t) \leq k_1 + 1$.

Proof. Suppose the claim is not true. Suppose $\lambda_G(s, t) = k_1$, and we need fewer than k_1 guards in G , say $k_1 - 1$ guards are enough to separate s from t in G . But then this will also be a valid strategy in G_1 , contradicting the induction hypothesis from which we know that $d_{G_1}(s, t) = \lambda_{G_1}(s, t) = k_1$. Suppose $\lambda_G(s, t) = k_1 + 1$, and we need fewer than $k_1 + 1$ guards in G , say k_1 guards are enough to separate s from t in G . If $\lambda_G(s, t)$ has more than one vertex of G_2 , then it will contradict the induction hypothesis from which we know that $\lambda_G(s, t) = k_1$. If it does not have any vertex of G_2 then we can replace this separator with the union of static (s, t) separator of G_1

and the vertex x . And if it contains exactly one vertex of G_2 , then that vertex can be replaced by one of the terminals x or y and it will still be a valid static (s, t) separator in G . Observe that, to separate s and t in G , at least one guard must be on one of the paths of type $s \rightarrow x \rightarrow$ some vertices of $G_2 \rightarrow y \rightarrow t$. If not then there is no guard which can block x, y , and the vertices of G_2 and so it will not be a valid separator. Additionally, this vertex is not needed in the dynamic (s, t) separator of G_1 . Hence we can obtain a $k_1 - 1$ size dynamic (s, t) separator in G_1 by eliminating this vertex, contradicting the induction hypothesis. \triangleleft

Case 3: s is one of the terminals and $t \in G_1$ or $t \in G_2$; $s = x$ or $s = y$; $t \neq x_1, t \neq y_1$

Case 3A: The composition is series.

1. If $s = x$ and $t \in G_2$ or if $s = y$ and $t \in G_1$, then the vertex at the junction of the composition is a separator of size one.
2. If $s = x$ and $t \in G_1$ or $s = y$ and $t \in G_2$, then the static separator of s and t in G is the static separator of s and t in G_1 or the static separator of s and t in G_2 , of size k_1 or k_2 respectively.

\triangleright **Claim 14.** If $s = x$ and $t \in G_1$, then $d_G(s, t) = \lambda_G(s, t) = k_1$, while if $s = y$ and $t \in G_2$, then $d_G(s, t) = \lambda_G(s, t) = k_2$.

Proof. Consider the first statement and suppose the claim is not true. Then we need fewer than k_1 guards in G , say $k_1 - 1$ guards are enough to separate s from t in G . But then this will also be a valid strategy in G_1 , contradicting the induction hypothesis. The same argument works for the second statement as well. \triangleleft

Case 3B: The composition is parallel.

\triangleright **Claim 15.** $\lambda_G(s, t) \leq \lambda_{G_1}(s, t) + 1$, if $t \in G_1$ and $\lambda_G(s, t) \leq \lambda_{G_2}(s, t) + 1$, if $t \in G_2$. This argument in this case is analogous to Case 2B.

Case 4. s and t are both terminals.

Case 4A: The composition is series.

1. If $s = x; t = y$, then the vertex at the junction of the composition is a static separator of size one.
2. If $s = x; t = z$, where z denotes the vertex at the junction of the composition, then a static separator of s and t in G_1 is also a static separator of s and t in G .
3. If $s = z; t = y$, where z is the same as before, then a static separator of s and t in G_2 is also a static separator of s and t in G .

\triangleright **Claim 16.** If $s = x$ and $t = z$, then $d_G(s, t) = \lambda_G(s, t) = k_1$, while if $s = z$ and $t = y$, then $d_G(s, t) = \lambda_G(s, t) = k_2$.

Proof. Consider the first statement and suppose the claim is not true. Then we need fewer than k_1 guards in G , say $k_1 - 1$ guards are enough to separate s from t in G . But then this will also be a valid strategy in G_1 , contradicting the induction hypothesis. The same argument works for the second statement as well. \triangleleft

27:22 Romeo and Juliet Meeting in Forest like Regions

Case 4B: The composition is parallel, i.e. $s = x; t = y$. In this case, note that the size of the static separator of s and t in $G = \lambda_{G_1}(s, t) + \lambda_{G_2}(s, t)$. Indeed, any smaller subset would have either fewer than $\lambda_{G_1}(s, t)$ vertices in $G[V(G_1)]$ or fewer than $\lambda_{G_2}(s, t)$ vertices in $G[V(G_2)]$, implying the existence of an unblocked path from s to t via G_1 or G_2

▷ Claim 17. $d_G(s, t) = \lambda_G(s, t) = k_1 + k_2$.

Proof. Suppose the claim is not true. Then we need fewer than $k_1 + k_2$ guards in G , say $k_1 + k_2 - 1$ guards are enough to separate s from t in G . But then this will also be a valid strategy in G_1 (or G_2) with $< k_1$ (or $< k_2$) guards, contradicting the induction hypothesis. ◁

So, for all the cases $\lambda_G(s, t) = d_G(s, t)$, and this concludes our argument. ◀

New Characterizations of Core Imputations of Matching and b -Matching Games

Vijay V. Vazirani ✉

University of California, Irvine, CA, USA

Abstract

We give new characterizations of core imputations for the following games:

1. The assignment game.
2. Concurrent games, i.e., general graph matching games having non-empty core.
3. The unconstrained bipartite b -matching game (edges can be matched multiple times).
4. The constrained bipartite b -matching game (edges can be matched at most once).

The classic paper of Shapley and Shubik [11] showed that core imputations of the assignment game are precisely optimal solutions to the dual of the LP-relaxation of the game. Building on this, Deng et al. [5] gave a general framework which yields analogous characterizations for several fundamental combinatorial games. Interestingly enough, their framework does not apply to the last two games stated above. In turn, we show that some of the core imputations of these games correspond to optimal dual solutions and others do not. This leads to the tantalizing question of understanding the origins of the latter.

We also present new characterizations of the profits accrued by agents and teams in core imputations of the first two games. Our characterization for the first game is stronger than that for the second; the underlying reason is that the characterization of vertices of the Birkhoff polytope is stronger than that of the Balinski polytope.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases LP-duality theory, cooperative game theory, core of a game, assignment game, general graph matching game, bipartite b -matching game

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.28

Related Version *Full Version:* <https://arxiv.org/pdf/2202.00619.pdf>

Funding *Vijay V. Vazirani:* Supported in part by NSF grants CCF-1815901 and CCF-2230414.

Acknowledgements I wish to thank Hervé Moulin for asking the interesting question of extending results obtained for the assignment game to general graph matching games having a non-empty core. I also wish to thank Federico Echenique, Hervé Moulin and Thorben Trobst for several valuable discussions.

1 Introduction

The matching game forms one of the cornerstones of cooperative game theory and the *core* is a quintessential solution concept in this theory; the latter captures all possible ways of distributing the total worth of a game among individual agents in such a way that the grand coalition remains intact, i.e., a sub-coalition will not be able to generate more profits by itself and therefore has no incentive to secede from the grand coalition. The matching game can also be viewed as a matching market in which utilities of the agents are stated in monetary terms and side payments are allowed, i.e., it is a *transferable utility (TU) market*. For an extensive coverage of these notions, see the book by Moulin [9]. Due to space restrictions, we have not presented proofs in this version of the paper; for that, see the full version [12].



© Vijay V. Vazirani;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 28; pp. 28:1–28:13



Leibniz International Proceedings in Informatics

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

The classic paper of Shapley and Shubik [11] showed that the set of core imputations of the assignment game is the set of optimal solutions to the dual of the LP-relaxation of the maximum weight matching problem in the underlying graph. Among their other insights was a characterization of the two “antipodal” points – imputations which maximally favor one side of the bipartition¹ – in the core of this game. This in-depth understanding makes the assignment game a paradigmatic setting for studying the core; in turn, insights gained provide valuable guidance on profit-sharing in real-life situations.

Deng et al. [5] distilled the ideas underlying the Shapley-Shubik Theorem to obtain a general framework (see Section 4.1.1) which helps characterize the core of several games that are based on fundamental combinatorial optimization problems, including maximum flow in unit capacity networks both directed and undirected, maximum number of edge-disjoint s - t paths, maximum number of vertex-disjoint s - t paths, maximum number of disjoint arborescences rooted at a vertex r , and concurrent games (defined below).

In this paper, we study the core of the assignment game and some of its generalizations, including two versions of the bipartite b -matching game (Section 4); in the first version (Section 4.2) edges can be matched multiple number of times and in the second, edges can be matched at most once (Section 5). The intriguing aspect of the latter two games is that they don’t fall in framework of Deng et al.; see Section 4.1.1 for the reason. In turn, we show that some of the core imputations of these games correspond to optimal dual solutions and some not. This leads to a tantalizing question: is there a “mathematical structure” that produces the latter?

For the assignment game (Section 3), we start with the realization is that despite the in-depth work of Shapley and Shubik, and the passage of half a century, there are still basic questions about the core which have remained unexplored:

1. Do core imputations spread the profit more-or-less evenly or do they restrict them to certain well-chosen agents? If the latter, what characterizes these “chosen” agents?
2. By definition, under any core imputation, the sum of profits of two agents i and j is at least the profit they make by being matched, say w_{ij} . What characterizes pairs (i, j) for which this sum strictly exceed w_{ij} ?
3. How do core imputations behave in the presence of degeneracy?

An assignment game is said to be *degenerate* if the optimal assignment is not unique. Although Shapley and Shubik had mentioned this phenomenon, they brushed it away, claiming that “in the most common case” the optimal assignment will be unique, and if not, their suggestion was to perturb the edge weights to make the optimal assignment unique. However, this is far from satisfactory, since perturbing the weights destroys crucial information contained in the original instance and the outcome becomes a function of the vagaries of the randomness imposed on the instance.

The following broad idea helps answer all three questions. A well-known theorem in matching theory says that the LP-relaxation of the optimal assignment problem always has an integral optimal solution [8]. Therefore, the worth of the assignment game is given by the optimal objective function value of this LP. Next, the Shapley-Shubik Theorem says that the set of core imputations of this game are precisely the optimal solutions to the dual of this LP. These two facts naturally raise the question of viewing core imputations through the lens of complementarity; in turn, it leads to a resolution of all three questions.

¹ Much like the top and bottom elements in a lattice of stable matchings.

The following setting, taken from [6] and [2], vividly captures the issues underlying profit-sharing in an assignment game. Suppose a coed tennis club has sets U and V of women and men players, respectively, who can participate in an upcoming mixed doubles tournament. Assume $|U| = m$ and $|V| = n$, where m, n are arbitrary. Let $G = (U, V, E)$ be a bipartite graph whose vertices are the women and men players and an edge (i, j) represents the fact that agents $i \in U$ and $j \in V$ are eligible to participate as a mixed doubles team in the tournament. Let w be an edge-weight function for G , where $w_{ij} > 0$ represents the expected earnings if i and j do participate as a team in the tournament. The total worth of the game is the weight of a maximum weight matching in G .

Assume that the club picks such a matching for the tournament. The question is how to distribute the total profit among the agents – strong players, weak players and unmatched players – so that no subset of players feel they will be better off seceding and forming their own tennis club. We will use this setting to discuss the issues involved in the questions raised above.

Under core imputations, the profit allocated to an agent is a function of the value he/she brings to the various sub-coalitions he/she belongs to, i.e., it is consistent with his/her negotiating power. Indeed, it is well known that core imputations provide profound insights into the negotiating power of individuals and sub-coalitions, see [9]. The first question provides further insights into this issue. Our answer to this question is that the core rewards only *essential* agents, namely those who are matched by *every* maximum weight matching, see Theorem 10.

Our answer to the second question is quite counter-intuitive: we show that a pair of players (i, j) get overpaid by core allocations if and only if they are so incompetent, as a team, that they don't participate in any maximum weight matching! Since i and j are incompetent as a team, w_{ij} is small. On the other hand, a least one of i and j does team up with other agents in maximum weight matchings – if not, (i, j) would have been matched in a maximum weight matching. Therefore, the sum of the profits of i and j exceeds w_{ij} in at least one core imputation; this is shown in Theorem 15.

Our insight into degeneracy is that it treats teams and agents in totally different ways, see Section 3.4. Section 2 discusses past approaches to degeneracy.

Whereas the core of the assignment game is always non-empty, that of the general graph matching game can be empty. Deng et al. [5] showed that the core of this game is non-empty if and only if the weights of maximum weight integral and fractional matchings concur. For this reason, we have named such games as *concurrent games*. As stated above, their core imputations are precisely the set of optimal solutions to the dual LP.

In the full paper [12], we study the three questions, raised above, for concurrent games as well.

2 Related Works

An imputation in the core has to ensure that *each* of the exponentially many sub-coalitions is “happy” – clearly, that is a lot of constraints. As a result, the core is non-empty only for a handful of games, some of which are mentioned in the Introduction. A different kind of game, in which preferences are cardinal, is based on the stable matching problem defined by Gale and Shapley [7]. The only coalitions that matter in this game are ones formed by one agent from each side of the bipartition. A stable matching ensures that no such coalition has the incentive to secede and the set of such matchings constitute the core of this game.

Over the years, researchers have approached the phenomenon of degeneracy in the assignment game from directions that are different from ours. Nunez and Rafels [10], studied relationships between degeneracy and the dimension of the core. They defined an agent to be *active* if her profit is not constant across the various imputations in the core, and non-active otherwise. Clearly, this notion has much to do with the dimension of the core, e.g., it is easy to see that if all agents are non-active, the core must be zero-dimensional. They prove that if all agents are active, then the core is full dimensional if and only if the game is non-degenerate. Furthermore, if there are exactly two optimal matchings, then the core can have any dimension between 1 and $m - 1$, where m is the smaller of $|U|$ and $|V|$; clearly, m is an upper bound on the dimension.

In another work, Chambers and Echenique [3] study the following question: Given the entire set of optimal matchings of a game on $m = |U|$, $n = |V|$ agents, is there an $m \times n$ surplus matrix which has this set of optimal matchings. They give necessary and sufficient conditions for the existence of such a matrix.

3 The Core of the Assignment Game

In this section, we provide answers to the three questions, for assignment games, which were raised in the Introduction.

3.1 Definitions and Preliminary Facts

The *assignment game*, $G = (U, V, E)$, $w : E \rightarrow \mathcal{R}_+$, has been defined in the Introduction. We start by giving definitions needed to state the Shapley-Shubik Theorem.

► **Definition 1.** *The set of all players, $U \cup V$, is called the grand coalition. A subset of the players, $(S_u \cup S_v)$, with $S_u \subseteq U$ and $S_v \subseteq V$, is called a coalition or a sub-coalition.*

► **Definition 2.** *The worth of a coalition $(S_u \cup S_v)$ is defined to be the maximum profit that can be generated by teams within $(S_u \cup S_v)$ and is denoted by $p(S_u \cup S_v)$. Formally, $p(S_u \cup S_v)$ is the weight of a maximum weight matching in the graph G restricted to vertices in $(S_u \cup S_v)$ only. $p(U \cup V)$ is called the worth of the game. The characteristic function of the game is defined to be $p : 2^{U \cup V} \rightarrow \mathcal{R}_+$.*

► **Definition 3.** *An imputation² gives a way of dividing the worth of the game, $p(U \cup V)$, among the agents. It consists of two functions $u : U \rightarrow \mathcal{R}_+$ and $v : V \rightarrow \mathcal{R}_+$ such that $\sum_{i \in U} u(i) + \sum_{j \in V} v(j) = p(U \cup V)$.*

► **Definition 4.** *An imputation (u, v) is said to be in the core of the assignment game if for any coalition $(S_u \cup S_v)$, the total worth allocated to agents in the coalition is at least as large as the worth that they can generate by themselves, i.e., $\sum_{i \in S_u} u(i) + \sum_{j \in S_v} v(j) \geq p(S)$.*

We next describe the characterization of the core of the assignment game given by Shapley and Shubik [11]³.

² Some authors prefer to call this a pre-imputation, while using the term imputation when individual rationality is also satisfied.

³ Shapley and Shubik had described this game in the context of the housing market in which agents are of two types, buyers and sellers. They had shown that each imputation in the core of this game gives rise to unique prices for all the houses. In this paper we will present the assignment game in a variant of

As stated in Definition 2, the worth of the game, $G = (U, V, E)$, $w : E \rightarrow \mathcal{R}_+$, is the weight of a maximum weight matching in G . Linear program (1) gives the LP-relaxation of the problem of finding such a matching. In this program, variable x_{ij} indicates the extent to which edge (i, j) is picked in the solution. Matching theory tells us that this LP always has an integral optimal solution [8]; the latter is a maximum weight matching in G .

$$\begin{aligned}
\max \quad & \sum_{(i,j) \in E} w_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{(i,j) \in E} x_{ij} \leq 1 \quad \forall i \in U, \\
& \sum_{(i,j) \in E} x_{ij} \leq 1 \quad \forall j \in V, \\
& x_{ij} \geq 0 \quad \forall (i,j) \in E
\end{aligned} \tag{1}$$

Taking u_i and v_j to be the dual variables for the first and second constraints of (1), we obtain the dual LP:

$$\begin{aligned}
\min \quad & \sum_{i \in U} u_i + \sum_{j \in V} v_j \\
\text{s.t.} \quad & u_i + v_j \geq w_{ij} \quad \forall (i,j) \in E, \\
& u_i \geq 0 \quad \forall i \in U, \\
& v_j \geq 0 \quad \forall j \in V
\end{aligned} \tag{2}$$

► **Theorem 5** (Shapley and Shubik [11]). *The imputation (u, v) is in the core of the assignment game if and only if it is an optimal solution to the dual LP, (2).*

By Theorem 5, the core of the assignment game is a convex polyhedron. Shapley and Shubik shed further light on the structure of the core by showing that it has two special imputations which are furthest apart and so can be thought of as antipodal imputations. In the tennis club setup, one of these imputations maximizes the earnings of women players and the other maximizes the earnings of men players.

Finally, we state a fundamental fact about LP (1); its corollary will be used in a crucial way in Theorems 10 and 15.

► **Theorem 6** (Birkhoff [1]). *The vertices of the polytope defined by the constraints of LP (1) are 0/1 vectors, i.e., they are matchings in G .*

► **Corollary 7.** *Any fractional matching in a bipartite graph is a convex combination of integral matchings.*

3.2 The first question: Allocations made to agents by core imputations

► **Definition 8.** *A generic player in $U \cup V$ will be denoted by q . We will say that q is:*

1. essential if q is matched in every maximum weight matching in G .
2. viable if there is a maximum weight matching M such that q is matched in M and another, M' such that q is not matched in M' .
3. subpar if for every maximum weight matching M in G , q is not matched in M .

the tennis setting given in the Introduction; this will obviate the need to define “prices”, hence leading to simplicity.

► **Definition 9.** Let y be an imputation in the core. We will say that q gets paid in y if $y_q > 0$ and does not get paid otherwise. Furthermore, q is paid sometimes if there is at least one imputation in the core under which q gets paid, and it is never paid if it is not paid under every imputation.

► **Theorem 10.** For every player $q \in (U \cup V)$:

q is paid sometimes \iff q is essential

Theorem 10 is equivalent to the following. For every player $q \in (U \cup V)$:

q is never paid \iff q is not essential

Thus core imputations pay only essential players and each of them is paid in some core imputation. Since we have assumed that the weight of each edge is positive, so is the worth of the game, and all of it goes to essential players. Hence we get:

► **Corollary 11.** In the assignment game, the set of essential players is non-empty and in every core imputation, the entire worth of the game is distributed among essential players; moreover, each of them is paid in some core imputation.

► **Remark 12.** Theorem 5 and Corollary 11 are of much consequence.

1. Corollary 11 reveals the following surprising fact: the set of players who are allocated profits in a core imputation is *independent* of the set of teams that play.
2. The identification of these players, and the exact manner in which the total profit is divided among them, follows the negotiating process. In turn, this process identifies agents who play in *all* possible maximum weight matchings.
3. Perhaps the most remarkable aspect of Theorem 5 is that each possible outcome of this very real process is captured by an inanimate object, namely an optimal solution to the dual, LP (2).

By Corollary 11, core imputations reward only essential players. This raises the following question: Can't a non-essential player, say q , team up with another player, say p , and secede, by promising p almost all of the resulting profit? The answer is "No", because the dual (2) has the constraint $y_q + y_p \geq w_{qp}$. Therefore, if $y_q = 0$, $y_p \geq w_{qp}$, i.e., p will not gain by seceding together with q .

3.3 The second question: Allocations made to teams by core imputations

► **Definition 13.** By a mixed doubles team we mean an edge in G ; a generic one will be denoted as $e = (u, v)$. We will say that e is:

1. essential if e is matched in every maximum weight matching in G .
2. viable if there is a maximum weight matching M such that $e \in M$, and another, M' such that $e \notin M'$.
3. subpar if for every maximum weight matching M in G , $e \notin M$.

► **Definition 14.** Let y be an imputation in the core of the game. We will say that e is fairly paid in y if $y_u + y_v = w_e$ and it is overpaid if $y_u + y_v > w_e$ ⁴. Finally, we will say that e is always paid fairly if it is fairly paid in every imputation in the core.

⁴ Observe that by the first constraint of the dual LP (2), these are the only possibilities.

► **Theorem 15.** *For every team $e \in E$:*

$$e \text{ is always paid fairly} \iff e \text{ is viable or essential}$$

Negating both sides of the implication proved in Theorem 15, we get the following implication. For every team $e \in E$:

$$e \text{ is subpar} \iff e \text{ is sometimes overpaid}$$

Clearly, this statement is equivalent to the statement proved Theorem 15 and hence contains no new information. However, it provides a new viewpoint. These two equivalent statements yield the following assertion, which at first sight seems incongruous with what we desire from the notion of the core and the just manner in which it allocates profits:

Whereas viable and essential teams are always paid fairly, subpar teams are sometimes overpaid.

How can the core favor subpar teams over viable and essential teams? An explanation is provided in the Introduction, namely a subpar team (i, j) gets overpaid because i and j create worth by playing in competent teams with other players.

Finally, we observe that contrary to Corollary 11, which says that the set of essential players is non-empty, the set of essential teams may be empty.

3.4 The third question: Degeneracy

Next we use Theorems 10 and 15 to get insights into degeneracy. Clearly, if an assignment game is non-degenerate, then every team and every player is either always matched or always unmatched in the set of maximum weight matchings in G , i.e., there are no viable teams or players. Since viable teams and players arise due to degeneracy, in order to understand the phenomenon of degeneracy, we need to understand how viable teams and players behave with respect to core imputations; this is done in the next corollary.

► **Corollary 16.** *In the presence of degeneracy, imputations in the core of an assignment game treat:*

- *viable players in the same way as subpar players, namely they are never paid.*
- *viable teams in the same way as essential teams, namely they are always fairly paid.*

4 The Core of Bipartite b -Matching Games

In this section, we will define two versions of the *bipartite b -matching game* and we will study their core imputations; both versions generalize the assignment game.

4.1 Definitions and Preliminary Facts

As in the assignment game, let $G = (U, V, E)$, $w : E \rightarrow \mathcal{R}_+$ be the underlying bipartite graph and edge-weight function. Let function $b : U \cup V \rightarrow \mathbf{Z}_+$ give a bound on the number of times a vertex can be matched. Under the *unconstrained bipartite b -matching game*, each edge can be matched multiple number of times and under the *constrained bipartite b -matching game*, each edge can be matched at most once. Observe that even in the first version, limits imposed by b on vertices will impose limits on edges – thus edge (i, j) can be matched at most $\min\{b_i, b_j\}$ times.

The *worth* of a coalition $(S_u \cup S_v)$, with $S_u \subseteq U, S_v \subseteq V$, is the weight of a maximum weight b -matching in the graph G restricted to vertices in $(S_u \cup S_v)$ only; we will denote this by $p(S_u \cup S_v)$. Whether an edge can be matched at most once or more than once depends on the version of the problem we are dealing with. $p(U \cup V)$ is called the *worth of the game*. The *characteristic function* of the game is defined to be $p : 2^{U \cup V} \rightarrow \mathcal{R}_+$. Definitions 3 and 4, defining an imputation and the core, carry over unchanged from the assignment game.

The tennis setting, given in the Introduction, provides a vivid description of these two variants of the b -matching game as well. Let K denote the maximum b -value of a vertex and assume that the tennis club needs to enter mixed doubles teams into K tennis tournaments. In the first variant, a team can play in multiple tournaments and in the second version, a team can play in at most one tournament. In both cases, a player i can play in at most b_i tournaments. The goal of the tennis club is to maximize its profit over all the tournaments and hence picks a maximum weight b -matching in G . An imputation in the core gives a way of distributing the profit in such a way that no sub-coalition has an incentive to secede.

Linear program (3) gives the LP-relaxation of the problem of finding a maximum weight b -matching for the unconstrained version. In this program, variable x_{ij} indicates the extent to which edge (i, j) is picked in the solution; observe that there is no upper bound on the variables x_{ij} since an edge can be matched any number of times.

$$\begin{aligned}
 \max \quad & \sum_{(i,j) \in E} w_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{(i,j) \in E} x_{ij} \leq b_i \quad \forall i \in U, \\
 & \sum_{(i,j) \in E} x_{ij} \leq b_j \quad \forall j \in V, \\
 & x_{ij} \geq 0 \quad \forall (i, j) \in E
 \end{aligned} \tag{3}$$

Taking u_i and v_j to be the dual variables for the first and second constraints of (3), we obtain the dual LP:

$$\begin{aligned}
 \min \quad & \sum_{i \in U} b_i u_i + \sum_{j \in V} b_j v_j \\
 \text{s.t.} \quad & u_i + v_j \geq w_{ij} \quad \forall (i, j) \in E, \\
 & u_i \geq 0 \quad \forall i \in U, \\
 & v_j \geq 0 \quad \forall j \in V
 \end{aligned} \tag{4}$$

Linear program (5) gives the LP-relaxation of the problem of finding a maximum weight b -matching for the constrained version. Observe that in this program, variables x_{ij} are upper bounded by 1, since an edge can be matched at most once.

$$\begin{aligned}
 \max \quad & \sum_{(i,j) \in E} w_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{(i,j) \in E} x_{ij} \leq b_i \quad \forall i \in U, \\
 & \sum_{(i,j) \in E} x_{ij} \leq b_j \quad \forall j \in V, \\
 & x_{ij} \leq 1 \quad \forall (i, j) \in E, \\
 & x_{ij} \geq 0 \quad \forall (i, j) \in E
 \end{aligned} \tag{5}$$

► **Remark 17.** In both in LPs (3) and (5), the matrices of coefficients of the constraints are totally unimodular [8], and therefore both LPs always have integral optimal solutions.

Taking u_i , v_j and z_{ij} to be the dual variables for the first, second and third constraints of (5), we obtain the dual LP:

$$\begin{aligned}
\min \quad & \sum_{i \in U} b_i u_i + \sum_{j \in V} b_j v_j + \sum_{(i,j) \in E} z_{ij} \\
\text{s.t.} \quad & u_i + v_j + z_{ij} \geq w_{ij} \quad \forall (i,j) \in E, \\
& u_i \geq 0 \quad \forall i \in U, \\
& v_j \geq 0 \quad \forall j \in V, \\
& z_{ij} \geq 0 \quad \forall (i,j) \in E
\end{aligned} \tag{6}$$

4.1.1 The Framework of Deng et al. [5]

In this section, we present the framework of Deng et al. [5], which was mentioned in the Introduction, and point out why it does not apply to the two versions of the b -matching game. Let $T = \{1, \dots, n\}$ be the set of n agents of the game. Let $w \in \mathbb{R}_+^m$ be an m -dimensional non-negative real vector specifying the weights of certain objects; in the assignment game, the objects are edges of the underlying graph. Let A be an $n \times m$ matrix with 0/1 entries whose i^{th} row corresponds to agent $i \in T$. Let x be an m -dimensional vector of variables and $\mathbb{1}$ be the n -dimensional vector of all 1s. Assume that the worth of the game is given by the objective function value of following integer program.

$$\begin{aligned}
\max \quad & w \cdot x \\
\text{s.t.} \quad & Ax \leq \mathbb{1}, \\
& x \in \{0, 1\}
\end{aligned} \tag{7}$$

Moreover, for a sub-coalition, $T' \subseteq T$ assume that its worth is given by the integer program obtained by replacing A by A' in (7), where A' picks the set of rows corresponding to agents in T' . The LP-relaxation of (7) is:

$$\begin{aligned}
\max \quad & w \cdot x \\
\text{s.t.} \quad & Ax \leq \mathbb{1}, \\
& x \geq 0
\end{aligned} \tag{8}$$

Deng et al. proved that if LP (8) always has an integral optimal solution, then the set of core imputations of this game is exactly the set of optimal solutions to the dual of LP (8).

As stated in Remark 17, the matrices of coefficients of both LPs (3) and (5) are totally unimodular and therefore these LPs always have integral optimal solutions. However, they still don't fall in the above-stated framework because their right-hand-sides are b values of the vertices and not $\mathbb{1}$.

4.2 The Core of the Unconstrained Bipartite b -Matching Game

Let I denote an instance of this game and let $C(I)$ denote its set of core imputations. We will show in Theorem 18 that corresponding to every optimal solution to the dual LP (4), there is an imputation in $C(I)$. Let $D(I)$ denote the set of all such core imputations. Since $D(I) \neq \emptyset$, we get Corollary 19 stating that the core of this game is non-empty. Next, we will give an instance I such that $D(I) \subset C(I)$, i.e., unlike the assignment game, I has core imputations that don't correspond to optimal solutions to the dual LP.

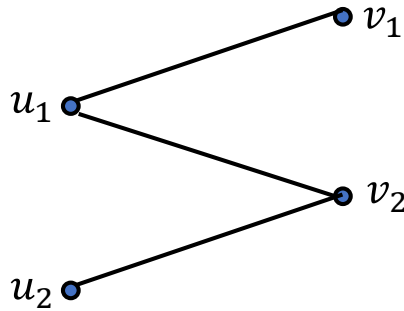
28:10 New Characterizations of Core Imputations

The correspondence between optimal solutions to the dual LP (4) and core imputations in $D(I)$ is as follows. Given an optimal solution (u, v) , define the profit allocation to $i \in U$ to be $\alpha_i = b_i \cdot u_i$ and that to $j \in V$ to be $\beta_j = b_j \cdot v_j$.

► **Theorem 18.** *The profit-sharing method (α, β) , which corresponds to an optimal solution (u, v) to the dual LP (4), is an imputation in the core of the unconstrained bipartite b -matching game.*

► **Corollary 19.** *The core of the unconstrained bipartite b -matching game is always non-empty.*

► **Remark 20.** Observe that the mapping given from optimal solutions to the dual LP (4) to core imputations in $D(I)$ is a bijection.



■ **Figure 1** The graph for Example 21.

► **Example 21.** For the bipartite b -matching game defined by the graph of Figure 1, let the b values be 2, 1, 2, 1 for u_1, u_2, v_1, v_2 , and let the edge weights be 1, 3, 1 for $(u_1, v_1), (u_1, v_2), (u_2, v_2)$.

In this section, we will view the game defined in Example 21 as an unconstrained bipartite b -matching game and will show that it has a set of core imputations which do not correspond to optimal dual solutions, i.e., they lie in $C(I) - D(I)$. The optimal matching picks edges $(u_1, v_1), (u_1, v_2)$ once each, for a total profit of 4. The unique optimal dual solution is 1, 0, 0, 2 for u_1, u_2, v_1, v_2 , and the corresponding core imputation is 2, 0, 0, 2.

Let $\alpha_1, \alpha_2, \beta_1, \beta_2$ be the profits allocated to u_1, u_2, v_1, v_2 . The solutions of the system of linear inequalities (9), for non-negative values of the variables, capture all possible core imputations, i.e., the set $C(I)$.

$$\begin{aligned}
 \alpha_1 + \beta_1 &\geq 2 \\
 \alpha_1 + \beta_2 &\geq 3 \\
 \alpha_1 + \beta_1 + \beta_2 &\geq 4 \\
 \alpha_2 + \beta_2 &\geq 1 \\
 \alpha_1 + \alpha_2 + \beta_2 &\geq 3 \\
 \alpha_1 + \alpha_2 + \beta_1 + \beta_2 &= 4
 \end{aligned} \tag{9}$$

On solving this system, we find that $\alpha_1, \alpha_2, \beta_1, \beta_2$ should be $1 + a, 0, 1 + c$, where a, b, c are non-negative and satisfy the system (10).

$$\begin{aligned} a + b &\geq 1 \\ a + c &\geq 1 \\ a + b + c &= 2 \end{aligned} \tag{10}$$

A fourth constraint, $b \leq 1$ follows from the last two in this system. The solution $a = 1, b = 0, c = 1$ gives the core imputation corresponding to the unique optimal dual solution; the rest give the remaining core imputations, e.g., the imputation $3, 0, 0, 1$.

For an arbitrary instance I , one can clearly capture all possible core imputations via an exponential sized system of inequalities of the type \geq , one corresponding to each coalition $(S_u \cup S_v)$; its r.h.s. will be $p(S_u \cup S_v)$ and its l.h.s. will be the sum of all variables denoting profits accrued to vertices in this coalition. Note that all the variables of this system will be constrained to be non-negative and it will have one equality corresponding to the worth of the grand coalition; the latter is the last equality in system (9).

The following question arises: is there a smaller system which accomplishes this task? We observe that it suffices to include in the system only those coalitions whose induced subgraph is connected. This is so because if the induced subgraph for coalition $(S_u \cup S_v)$ has two or more connected components, then the sum of the inequalities for the connected components yields the inequality for coalition $(S_u \cup S_v)$. In particular, if the underlying graph of instance I is sparse, this may lead to a much smaller system. Observe that the system (9), for Example 21, follows from this idea.

► Remark 22. Since for the unconstrained bipartite b -matching game, the optimal dual solutions don't capture all core imputations, the characterizations established in Theorems 10 and 15 for the assignment game, don't carry over. However, if one restricts to core imputations in the set $D(I)$ only, one can see that suitable modifications of these statements do hold.

5 The Core of the Constrained Bipartite b -Matching Game

Our results for this game are related to, though not identical with, those for the unconstrained version. In Theorem 23, we will show that corresponding to every optimal solution to the dual LP (6), there is a set of core imputations. This theorem yields Corollary 24 stating that the core of this game is also non-empty. Finally, we will give an instance which has core imputations that don't correspond to optimal solutions to the dual LP.

The corresponding to an optimal solution to the dual LP (4), (u, v, z) , we define a set of imputations as follows. For each edge (i, j) define two new variables c_{ij} and d_{ij} ; both are constrained to be non-negative. Furthermore, consider all possible ways of splitting z_{ij} into c_{ij} and d_{ij} , i.e., $z_{ij} = c_{ij} + d_{ij}$. Observe that if $x_{ij} = 0$ then $z_{ij} = 0$ and therefore $c_{ij} = d_{ij} = 0$. Define the profit allocation to $i \in U$ to be

$$\alpha_i = b_i \cdot u_i + \sum_{(i,j) \in E} c_{ij}$$

and that to $j \in V$ to be

$$\beta_j = b_j \cdot v_j + \sum_{(i,j) \in E} d_{ij}.$$

Taken over all possible ways of splitting all z_{ij} s, this gives a set of imputations.

► **Theorem 23.** *All profit-sharing methods (α, β) , which correspond to the optimal solution (u, v, z) to the dual LP (6), are imputations in the core of the constrained bipartite b -matching game.*

► **Corollary 24.** *The core of the constrained bipartite b -matching game is always non-empty.*

In this section, we will view the game defined in Example 21 as a constrained bipartite b -matching game and will again show that it has a set of core imputations which do not correspond to optimal dual solutions. The optimal matching picks edges $(u_1, v_1), (u_1, v_2)$ once each, for a total profit of 4. Unlike the unconstrained case, this time, the optimal dual is not unique. The optimal dual solutions are given by $1, 0, 0, 2 - a$, for vertices u_1, u_2, v_1, v_2 , and $0, a, 0$ for edges $(u_1, v_1), (u_1, v_2), (u_2, v_2)$, where $a \in [0, 1]$. The corresponding core imputations are $3 - b, 0, 0, 1 + b$, for the four vertices u_1, u_2, v_1, v_2 , where $b \in [0, 1]$.

As in the unconstrained case, let $\alpha_1, \alpha_2, \beta_1, \beta_2$ be the profits allocated to u_1, u_2, v_1, v_2 . This time, the system of linear inequalities whose solutions capture all possible core imputations is given by system (9) after replacing the first inequality by

$$\alpha_1 + \beta_1 \geq 1.$$

This is so because edge (u_1, v_1) can be matched twice under the unconstrained bipartite b -matching game, but only once under the constrained version. As before, non-negativity is imposed on all these variables. On solving this system, we find that $\alpha_1, \alpha_2, \beta_1, \beta_2$ should be $1, 0, b, 1 + c$, where a, b, c are non-negative and satisfy the system (11).

$$\begin{aligned} a + b &\geq 1 \\ a + c &\geq 2 \\ a + b + c &= 3 \end{aligned} \tag{11}$$

Solutions of this system which do not correspond to dual solutions include $1, 0, 0, 3$ and $0, 0, 1, 3$. Observe that neither of these is a core imputation for the unconstrained bipartite b -matching game. The method given in Section 4.2, for finding a smaller system, holds for this case as well and so does Remark 22.

► **Remark 25.** In the assignment game, core imputations were precisely optimal dual solutions. On the other hand, in both versions of the bipartite b -matching game, core imputations are obtained from optimal dual solutions via specific operations. As stated in Remark 20, for the unconstrained version, there is a bijection between optimal dual solutions and core imputations in $D(I)$. In contrast, for the constrained version, the set of imputations corresponding to optimal dual solutions may not be disjoint.

Let us illustrate the last point of Remark 25 via Example 21. Consider the two optimal dual solutions obtained by setting $a = 0$ and $a = 1$, namely $1, 0, 0, 2$, for vertices u_1, u_2, v_1, v_2 , and $0, 0, 0$ for edges $(u_1, v_1), (u_1, v_2), (u_2, v_2)$; and $1, 0, 0, 1$, for vertices u_1, u_2, v_1, v_2 , and $0, 1, 0$ for edges $(u_1, v_1), (u_1, v_2), (u_2, v_2)$. Both these optimal duals yield the core imputation assigning profits of $2, 0, 0, 2$ for u_1, u_2, v_1, v_2 .

6 Discussion

Our most important open question is to shed light on the origins of core imputations, for the two bipartite b -matching games, which do not correspond to optimal dual solutions. Is there a “mathematical structure” that produces them? A related question is to determine

the complexity of the following question for these two games: Given an imputation for a game, decide if it belongs to the core. We believe this question should be co-NP-complete. On the other hand, the following question is clearly in P: Given an imputation for a game I , decide if it lies in $D(I)$.

As stated in Section 3.1, for the assignment game, Shapley and Shubik were able to characterize “antipodal” points in the core. An analogous understanding of the core of the general graph matching games having non-empty core will be desirable.

For the assignment game, Demange, Gale and Sotomayor [4] give an auction-based procedure to obtain a core imputation; it turns out to be optimal for the side that proposes, as was the case for the deferred acceptance algorithm of Gale and Shapley [7] for stable matching. Is there an analogous procedure for obtaining an imputation in the core of the general graph matching games having non-empty core?

References

- 1 Garrett Birkhoff. Three observations on linear algebra. *Univ. Nac. Tacuman, Rev. Ser. A*, 5:147–151, 1946.
- 2 Péter Biró, Walter Kern, and Daniël Paulusma. Computing solutions for matching games. *International journal of game theory*, 41(1):75–90, 2012.
- 3 Christopher P Chambers and Federico Echenique. The core matchings of markets with transfers. *American Economic Journal: Microeconomics*, 7(1):144–64, 2015.
- 4 Gabrielle Demange, David Gale, and Marilda Sotomayor. Multi-item auctions. *Journal of political economy*, 94(4):863–872, 1986.
- 5 Xiaotie Deng, Toshihide Ibaraki, and Hiroshi Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Mathematics of Operations Research*, 24(3):751–766, 1999.
- 6 Kimmo Eriksson and Johan Karlander. Stable outcomes of the roommate game with transferable utility. *International Journal of Game Theory*, 29(4):555–569, 2001.
- 7 David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
- 8 L. Lovász and M.D. Plummer. *Matching Theory*. North-Holland, Amsterdam–New York, 1986.
- 9 Hervé Moulin. *Cooperative microeconomics: a game-theoretic introduction*, volume 313. Princeton University Press, 2014.
- 10 Marina Núñez and Carles Rafels. On the dimension of the core of the assignment game. *Games and Economic Behavior*, 64(1):290–302, 2008.
- 11 Lloyd S Shapley and Martin Shubik. The assignment game I: The core. *International Journal of Game Theory*, 1(1):111–130, 1971.
- 12 Vijay V Vazirani. New characterizations of core imputations of matching and b -matching games. *arXiv preprint*, 2022. [arXiv:2202.00619](https://arxiv.org/abs/2202.00619).

Algorithms and Hardness Results for Computing Cores of Markov Chains

Ali Ahmadi  

Hong Kong University of Science and Technology (HKUST), China

Krishnendu Chatterjee  



Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Amir Kafshdar Goharshady  

Hong Kong University of Science and Technology (HKUST), China

Tobias Meggendorfer  

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Roodabeh Safavi  

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Hong Kong University of Science and Technology (HKUST), China

Đorđe Žikelić  

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

Given a Markov chain $M = (V, v_0, \delta)$, with state space V and a starting state v_0 , and a probability threshold ϵ , an ϵ -core is a subset C of states that is left with probability at most ϵ . More formally, $C \subseteq V$ is an ϵ -core, iff $\mathbb{P}[\text{reach}(V \setminus C)] \leq \epsilon$. Cores have been applied in a wide variety of verification problems over Markov chains, Markov decision processes, and probabilistic programs, as a means of discarding uninteresting and low-probability parts of a probabilistic system and instead being able to focus on the states that are likely to be encountered in a real-world run. In this work, we focus on the problem of computing a minimal ϵ -core in a Markov chain. Our contributions include both negative and positive results: (i) We show that the decision problem on the existence of an ϵ -core of a given size is NP-complete. This solves an open problem posed in [26]. We additionally show that the problem remains NP-complete even when limited to acyclic Markov chains with bounded maximal vertex degree; (ii) We provide a polynomial time algorithm for computing a minimal ϵ -core on Markov chains over control-flow graphs of structured programs. A straightforward combination of our algorithm with standard branch prediction techniques allows one to apply the idea of cores to find a subset of program lines that are left with low probability and then focus any desired static analysis on this core subset.

2012 ACM Subject Classification Software and its engineering \rightarrow Formal software verification; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases Markov Chains, Cores, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.29

Funding The research was partially supported by the Hong Kong Research Grants Council ECS Project No. 26208122, ERC CoG 863818 (FoRM-SMART), the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 665385, HKUST-Kaisa Joint Research Institute Project Grant HKJRI3A-055 and HKUST Startup Grant R9272. Ali Ahmadi and Roodabeh Safavi were interns at HKUST. Author names appear in alphabetical order.



© Ali Ahmadi, Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, Roodabeh Safavi, and Đorđe Žikelić; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 29; pp. 29:1–29:20



Leibniz International Proceedings in Informatics

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

1 Introduction

Markov Chains. Discrete-time Markov chains (MCs) are arguably the most classical and standard mathematical formalism for modelling randomness in discrete-time probabilistic systems [19]. In a nutshell, Markov chains comprise a set of states and a transition function, assigning to each state a distribution over successors. The system evolves by repeatedly drawing a successor state from the transition distribution of the current state. This can, for example, model communication over a lossy channel, a queuing network, or populations of predator and prey which grow and interact randomly. Indeed, Markov chains are an important tool in many areas, such as computer science [4], biology [30], epidemiology [21], and chemistry [20], to name a few. As such, analyzing MCs is an important question, and a central component of many modern model checkers, such as Prism [27] and Storm [17].

Markov Chains over Control-flow Graphs. An interesting family of MCs in verification are those that are defined over the control-flow graph (CFG) of a program [1], i.e. MCs in which the underlying graph, ignoring the probabilities, is a CFG. Such MCs can serve as formal semantics for simple probabilistic programs. They also appear naturally as a result of applying (statistical) branch prediction to a non-probabilistic program [33].

Markov Decision Processes. Extending MCs with non-determinism leads to the standard notion of Markov decision processes (MDPs) [31], which is the most common formalism for systems that exhibit both probabilistic and non-deterministic behavior. The non-determinism can model a wide variety of real-world phenomena such as the behavior of a controller, an unknown set of inputs, or simply abstractions made to ease the verification of the underlying system. Analysis of MDPs is a central topic in formal verification and model checking, see e.g. [2, 3, 4, 6, 10, 26].

Probabilistic Programs. Extending classical programs with the ability of generating random numbers by sampling from pre-defined distributions leads to the framework of probabilistic programs. Such programs have a wide variety of use-cases including randomized algorithms [18, 29], stochastic network protocols [34], blockchain protocols and smart contracts [11, 13, 14], and robotics [23, 25]. Hence, they have been widely studied by the programming languages and verification communities [5, 24, 28, 7, 8, 22, 35, 15]. Probabilistic programs often have variables that can take integer or real values and their formal semantics are usually defined by an infinite-state MC, in absence of non-determinism, or an infinite-state MDP, in its presence [7, 36, 9].

Cores. Fixing a probability threshold $\epsilon \in (0, 1)$, an ϵ -core of a probabilistic system is a set of states that is left by a random run of the system with probability at most ϵ . Intuitively, when analyzing a large probabilistic system, one is interested in finding a core that is left with very low probability and then ignore the set of states that are outside this core, since it is unlikely that a real-world run of the system visits them. If the core happens to be much smaller than the original system, this simple idea can lead to huge improvements in the runtime of formal analyses. For example, suppose that we have applied branch prediction to a huge program and hence have probabilities associated to every transition in its CFG. By

finding a small core, we can focus any static analysis on a set of lines of the program that actually matter, i.e. a set of lines that is left with low probability, and ignore the lines that are rarely or never encountered.

The concept of cores was introduced in [26] for the quantitative analysis of MCs and MDPs. Considering classical objectives such as mean-payoff, discounted sums of rewards, and hitting probabilities, [26] provided a partial-exploration framework that efficiently finds cores in large MDPs and then uses them to obtain not only a faster analysis, but also rigorous error bounds. An equivalent notion, called stochastic invariants, was introduced in [16] in the context of probabilistic programs and used in [12] to obtain quantitative bounds on their probability of termination.

Hardness of Computing a Core. Based on the discussion above, it is natural to aim for algorithms that find the smallest possible core in a probabilistic system, be it an MC, an MDP or a probabilistic program. In the case of probabilistic programs, any non-trivial formalization of this problem is undecidable as a direct result of the well-known Rice’s theorem [32]. In the case of finite-state MDPs, given a threshold $\epsilon \in (0, 1)$ and an integer k , deciding whether the MDP has an ϵ -core of size k is an NP-complete problem [26]. However, the same problem was open when it comes to MCs [26], with neither efficient PTIME algorithms nor NP-hardness results provided so far.

Our Contribution. In this work, we consider the problem of finding an optimal ϵ -core in a finite-state discrete-time Markov chain and its decision variant, i.e. deciding whether an ϵ -core of size k exists. We obtain both hardness results and efficient algorithms:

- In Section 3, we prove that the decision problem is NP-complete. This settles the complexity for MCs and answers the open problem posed in [26]. We also show that the problem remains NP-hard even when limited to acyclic MCs with bounded degree.
- We then focus on positive results and provide a PTIME algorithm in Section 4 that is applicable to MCs over control-flow graphs.

In summary, our results show that computing cores in MCs is NP-hard in general and even over the very limited family of acyclic MCs with bounded degrees. Hence, in practice, cores should be computed using partial-exploration algorithms, as in [26], or other heuristics with no theoretical guarantees. However, in certain important use-cases, such as MCs over CFGs, we can compute an optimal core in PTIME. Notably, this is applicable to the problem of finding the core lines of a program given branch prediction data.

2 Preliminaries

We start by recalling the definitions of Markov chains and cores and fixing the notation that is used throughout this work.

Discrete Probability Distributions. Given a finite set X , a *probability distribution* over X is a function $\delta : X \rightarrow [0, 1]$ which satisfies the condition $\sum_{x \in X} \delta(x) = 1$. We use $\mathcal{D}(X)$ to denote the set of all probability distributions over X .

Markov Chains (MCs). A finite state *Markov chain* $M = (V, v_0, \delta)$ is an ordered triple consisting of a finite set of *states/vertices* V , a designated *initial state* $v_0 \in V$ and a *probabilistic transition function* $\delta : V \rightarrow \mathcal{D}(V)$, which, to each state in V assigns a probability distribution over its successor states. We define the set of *edges* of M as

$$E = \{(v, v') \in V \times V \mid \delta(v)(v') > 0\}.$$

Paths in Markov Chains. An infinite path in a Markov chain $M = (V, v_0, \delta)$ is an infinite sequence of states $\rho = (v_0, v_1, v_2, \dots)$ such that $\delta(v_i)(v_{i+1}) > 0$ holds for each $i \in \mathbb{N}$. Note that we require each infinite path to start in the initial state v_0 of the chain.

A Markov chain M admits a *probability measure* \mathbb{P}^M over the set of all infinite paths in the Markov chain [31]. For each measurable set $O \subseteq V^\omega$ of infinite paths in M , we use $\mathbb{P}^M[O]$ to denote the probability of a random infinite path in the Markov chain being an element of the set O . In particular, if $T \subseteq V$ is a set of states, we use $\text{reach } T$ to denote the set of all infinite paths in the Markov chain that visit at least one state in T , and use $\mathbb{P}^M[\text{reach } T]$ to denote the probability of a random infinite path in the Markov chain reaching a vertex in T . When the Markov chain is clear from the context, we abbreviate the notation to \mathbb{P} , $\mathbb{P}[O]$ and $\mathbb{P}[\text{reach } T]$.

Cores in Markov Chains. We now define the concept of cores in Markov chains which are the central object of study in this work. The notion of a core was originally defined in [26]. Given a probability threshold $\epsilon \in [0, 1]$, an ϵ -*core* in a Markov chain $M = (V, v_0, \delta)$ is a set of states $C \subseteq V$ such that a random infinite path in the Markov chain exits C with probability at most ϵ . More formally,

$$\mathbb{P}[\text{reach}(V \setminus C)] \leq \epsilon.$$

For a natural number $k \in \mathbb{N}$, we say that $C \subseteq V$ is an (ϵ, k) -core if it is an ϵ -core of size at most k , i.e. $|C| \leq k$.

While the main negative result of this work is a proof of NP-completeness of the decision problem on whether a Markov chain contains an ϵ -core of at most a given size for $\epsilon \in (0, 1)$, we also strengthen this result by showing that the problem remains NP-complete even on a restricted class of Markov chains. We now formally define these restrictions.

Acyclic Markov Chains with Bounded Degree. A Markov chain $M = (V, v_0, \delta)$ naturally induces a directed graph $G_M = (V, E)$, where the vertex and the edge sets of the graph are defined by the set of states and the set of edges of the Markov chain. A Markov chain M is said to be *acyclic* if its induced graph G_M is acyclic, with the exception of a self-loop at one vertex. Note that the graph G_V cannot be completely acyclic, since it is a finite, directed graph and every state in a Markov chain must have at least one successor state for the probabilities of its outgoing edges to sum up to 1. Thus, a Markov chain must contain a proper cycle or a probability 1 self-loop. Therefore, acyclic Markov chains can be viewed as Markov chains whose induced graphs are “closest” to being acyclic, with the exception of a single probability 1 self-loop.

For a vertex/state $v \in V$, we define its *outdegree* to be the number of edges whose source vertex is v , i.e. $\text{out}(v) = |\{u \in V \mid (v, u) \in E\}|$. Similarly, the *indegree* of v is defined as the number of edges whose target vertex is v , i.e. $\text{in}(v) = |\{u \in V \mid (u, v) \in E\}|$. Together, we

define the *degree* of a vertex v to be the total number of edges that are incident to v , i.e. $\deg(v) = \text{out}(v) + \text{in}(v)$. Finally, the *maximal vertex degree* of a Markov chain M is defined to be maximal degree of all its vertices, i.e. $\max_{v \in V} \deg(v)$.

In this work, we consider Markov chains which are both acyclic and have their maximal degree bounded by a constant.

3 Hardness Results

In this section, we show that the problem of computing the size of a minimal ϵ -core in a Markov chain is NP-hard for any non-trivial value of the probability threshold ϵ , i.e. any ϵ that is not equal to 0 or 1. Formally, given $\epsilon \in (0, 1)$, we prove in Theorem 1 that the problem of deciding whether a Markov chain M contains an (ϵ, k) -core for a given Markov chain M and a core size k is NP-complete. This implies that the problem of deciding whether a Markov chain contains an ϵ -core of size exactly k is also NP-complete, since one may always enlarge an (ϵ, k) -core in order to obtain an ϵ -core that contains exactly k states and so the two problems are immediately reducible to each other.

In [26], the authors proved NP-completeness of the problem for MDPs. However, it was hitherto not known whether computing cores of given size could be made more efficient in the case of Markov chains. This was left as an open problem in [26, Remark 3.7]. Our Theorem 1 solves the open problem of [26] and answers the posed question negatively. We conclude this section with Theorem 8, which shows that the problem of deciding the existence of a core of at most given size remains NP-complete even if we restrict it to acyclic Markov chains whose maximal vertex degree is at most 3.

► **Theorem 1** (Hardness of Finding a Core in an MC). *For an $\epsilon \in (0, 1)$, let CORE_ϵ be the language defined as*

$$\left\{ (M, k) \mid M \text{ is a Markov chain that contains an } (\epsilon, k)\text{-core} \right\}.$$

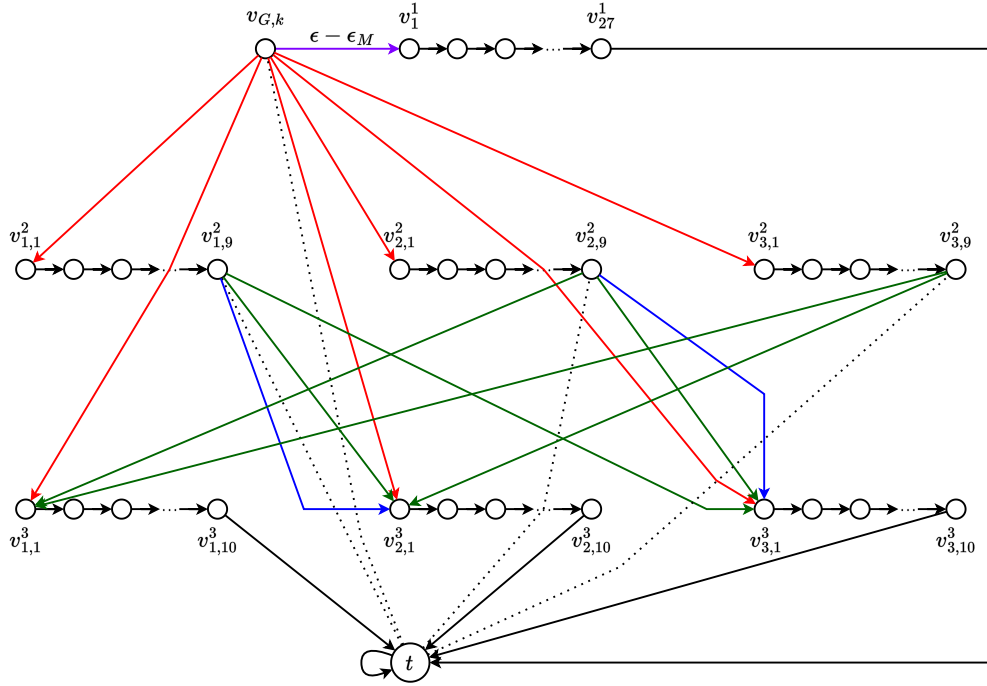
Then, CORE_ϵ is NP-complete for any $\epsilon \in (0, 1)$.

Proof. To prove that CORE_ϵ is contained in NP, note that the probability of reaching a given set of states in a Markov chain can be computed in polynomial time by reduction to a system of linear equations [4]. Hence, an (ϵ, k) -core can serve as its own witness of linear size. Thus, CORE_ϵ is in NP.

To prove that CORE_ϵ is NP-hard, we show a reduction from the VERTEX-COVERproblem which is a classical example of an NP-complete problem. Recall that, given an undirected graph $G = (V, E)$ and $k \in \mathbb{N}$, the VERTEX-COVERproblem is concerned with deciding if there exists a set of vertices $K \subseteq V$ of size k such that each edge in E is incident to at least one vertex in K .

Fix $\epsilon \in (0, 1)$ and consider an instance $(G = (V, E), k)$ of the VERTEX-COVERproblem where $n = |V| > \frac{1}{\epsilon}$ and $n \geq 4$. In order to obtain our reduction, we construct an instance $(M_{G,k}, k_{G,k})$ of the CORE_ϵ problem where the Markov chain $M_{G,k}$ and natural number $k_{G,k}$ are both polynomial in the size of G and k and such that $(G, k) \in \text{VERTEX-COVER}$ if and only if $(M_{G,k}, k_{G,k}) \in \text{CORE}_\epsilon$.

In the sequel, we say that a sequence of vertices u_1, \dots, u_s in a Markov chain (M, v, δ) form a *chain* of length s , if $\delta(u_i)(u_{i+1}) = 1$ for each $1 \leq i \leq s - 1$.



■ **Figure 1** The figure depicts an example of a Markov chain $V_{G,k}$ constructed in the proof of Theorem 1 for a graph $G = (V, E)$ with $V = \{v_1, v_2, v_3\}$ and $E = \{(v_1, v_2), (v_2, v_3)\}$, and for an arbitrary value of k . Let $n = |V| = 3$. The states of the Markov chain $V_{G,k}$ are visually ordered in four layers. The first layer consists of the initial state $v_{G,k}$ and a chain of length $n^3 = 27$. The second layer consists of $n = 3$ chains, each of length $3 \cdot n = 9$. The third layer also consists of $n = 3$ chains, each of length $3 \cdot n + 1 = 10$. Finally, the fourth layer consists of a single state t . Edges induced by the probabilistic transition function are indicated by directed lines between states. Chain edges and all other edges of probability 1 are colored in black. Edges of probability $p_1 = \frac{1}{n^3}$ are colored in red, edges of probability $p_2 = \frac{1}{n^{10}}$ in green and edges of probability $p_3 = \frac{1}{n^{30}}$ in blue. The edge of probability $\epsilon - \epsilon_M$ from the initial state $v_{G,k}$ to the first state v_1^1 of the chain in the first layer is colored in purple. Finally, dotted edges depict the remaining edges to the state t in the fourth layer, which are introduced in order to ensure the probabilities of outgoing edges from each state sum up to 1.

Construction of $k_{G,k}$. We set $k_{G,k} := 2 + 3 \cdot n^2 + k$.

Construction of $M_{G,k}$. Fix an enumeration $V = \{v_1, \dots, v_n\}$ of vertices in G . We construct the Markov chain $M_{G,k} = (V_{G,k}, v_{G,k}, \delta_{G,k})$ as follows: The state set $V_{G,k}$ consists of 4 disjoint subsets which we refer to as *layers* (see Figure 1 for a visualization of layers):

- The first layer consists of the initial state $v_{G,k}$ of the Markov chain $M_{G,k}$, as well as of n^3 states $v_1^1, \dots, v_{n^3}^1$ that form a chain of length n^3 .
- The second layer consists of n chains where each chain has length $3 \cdot n$. For each $1 \leq i \leq n$ and $1 \leq j \leq 3 \cdot n$, we use $v_{i,j}^2$ to denote the j -th state along the i -th chain.
- The third layer also consists of n chains where each chain has length $3 \cdot n + 1$. For each $1 \leq i \leq n$ and $1 \leq j \leq 3 \cdot n + 1$, we use $v_{i,j}^3$ to denote the j -th state along the i -th chain.
- The fourth layer consists of a single state t .

In addition to transition probabilities defined by the chains specified above, the probabilistic transition relation $\delta_{G,k}$ is defined as follows:

- The last vertex in the chain in the first layer is connected to the vertex t in the fourth layer by an edge of probability 1, i.e. $\delta_{G,k}(v_{n^3}^1)(t) = 1$.
- The initial state $v_{G,k}$ is connected to the first state of each chain in the second layer and each chain in the third layer by an edge of probability $p_1 = \frac{1}{n^3}$, i.e. $\delta_{G,k}(v_{G,k})(v_{i,1}^2) = \delta_{G,k}(v_{G,k})(v_{i,1}^3) = \frac{1}{n^3}$ for each $1 \leq i \leq n$.
- For each $1 \leq i \leq n$, the last state in the i -th chain in the second layer is connected to the first states of all but the i -th chain in the third layer by an edge of probability $p_2 = \frac{1}{n^{10}}$, i.e. $\delta_{G,k}(v_{i,3-n}^2)(v_{j,1}^3) = \frac{1}{n^{10}}$ for each $j \neq i$.
- For each edge (v_i, v_j) in graph G with $i < j$, the last state of the i -th chain in the second layer and the first state of the j -th chain in the third layer are connected by an edge of probability $p_3 = \frac{1}{n^{50}}$, i.e. $\delta_{G,k}(v_{i,3-n}^2)(v_{j,1}^3) = \frac{1}{n^{50}}$.
- The initial state $v_{G,k}$ is connected to the first state v_1^1 of the chain of length n^3 in the first layer by an edge of probability $\epsilon - \epsilon_M$ where $\epsilon_M = n \cdot p_1 + (n - k) \cdot (n - k - 1) \cdot p_1 \cdot p_2$, i.e. $\delta_{G,k}(v_{G,k})(v_1^1) = \epsilon - \epsilon_M$. By our choices of p_1 and p_2 , one can see that $\epsilon_M \leq \frac{2}{n^2} \leq \frac{1}{n} < \epsilon$ since we assume that $n > \frac{1}{\epsilon} > 1$.
- Vertex t in the fourth layer has a probability 1 self-loop, i.e. $\delta_{G,k}(t)(t) = 1$.
- Finally, for each vertex for which the probabilities of outgoing edges do not sum up to 1, we introduce an additional edge to t of the probability needed to make this sum equal to 1. This ensures that our construction of $M_{G,k}$ indeed yields a Markov chain.

$M_{G,k}$ contains $n^3 + 6 \cdot n^2 + n + 2$ vertices and transition probabilities are of size polynomial in n , hence the size of $M_{G,k}$ is polynomial in the size of G and k .

Correctness of reduction. To prove correctness of the reduction, it remains to show that $(G, k) \in \text{VERTEX-COVER}$ if and only if $(M_{G,k}, k_{G,k}) \in \text{CORE}_\epsilon$.

First, suppose that $(G, k) \in \text{VERTEX-COVER}$ and let $K \subseteq V$ be a vertex cover of size k in G . Then, letting

$$C = \{v_{G,k}, t\} \cup \{v_{i,1}^3, \dots, v_{i,3n+1}^2 \mid v_i \in K\} \cup \{v_{i,1}^2, \dots, v_{i,3n}^3 \mid v_i \notin K\}$$

defines an $(\epsilon, k_{G,k})$ -core in $M_{G,k}$. Indeed, we have that $|C| = 2 + k \cdot (3 \cdot n + 1) + (n - k) \cdot 3 \cdot n = 2 + 3 \cdot n^2 + k = k_{G,k}$ and due to our choice of ϵ_M one can verify by inspection that the probability of leaving C in $M_{G,k}$ is exactly equal to ϵ .

Second, we prove that $(M_{G,k}, k_{G,k}) \in \text{CORE}_\epsilon$ implies $(G, k) \in \text{VERTEX-COVER}$ by making a series of observations which will together imply the claim. Define a Markov chain $\bar{M}_{G,k}$ from $M_{G,k}$ by removing the chain $v_1^1, \dots, v_{n^3}^1$ from the first layer and increasing the probability of the transition from the initial state $v_{G,k}$ to the state t in the fourth layer by $\epsilon - \epsilon_M$.

► **Observation 2.** $M_{G,k}$ contains an $(\epsilon, k_{G,k})$ -core if and only if $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core. Thus, it suffices to show that $(\bar{M}_{G,k}, k_{G,k}) \in \text{CORE}_{\epsilon_M}$ implies $(G, k) \in \text{VERTEX-COVER}$.

To see this, note that $n^3 > k_{G,k} = 2 + 3 \cdot n^2 + k$ as we assume that $n \geq 4$ and $k \leq n$. Hence, no $(\epsilon, k_{G,k})$ -core in $M_{G,k}$ can contain the whole chain $v_1^1, \dots, v_{n^3}^1$ and the probability of visiting a state in $v_1^1, \dots, v_{n^3}^1$ that is not contained in a $(\epsilon, k_{G,k})$ -core is always equal to $\epsilon - \epsilon_M$. Hence, any $(\epsilon, k_{G,k})$ -core in $M_{G,k}$ can be modified into an $(\epsilon, k_{G,k})$ -core that contains

no state in the chain $v_1^1, \dots, v_{n^3}^1$ by removing all states from the chain. The set of states contained in this new core would give rise to an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$. Conversely, a set of states that form an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$ also form an $(\epsilon, k_{G,k})$ -core in $M_{G,k}$. This proves Observation 2.

► **Observation 3.** *If C is an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$, then removing states of all chains in the second and all chains in the third layer that are not entirely contained in C also gives rise to an $(\epsilon_M, k_{G,k})$ -core.*

To see this, observe that each edge between two successive states in a chain has probability 1, therefore if the whole chain is not contained in C then one may remove all states of that chain from C without increasing the probability of a random infinite path in $\bar{M}_{G,k}$ leaving C . Hence, removal of such states from C gives rise to an $(\epsilon_M, k_{G,k})$ -core.

► **Observation 4.** *If $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core, then it also contains an $(\epsilon_M, k_{G,k})$ -core that consists of the initial state $v_{G,k}$, the state t and all states of exactly n chains. Furthermore, at most k of these chains are contained in the third layer.*

Consider an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$. The initial state $v_{G,k}$ is contained in the core as otherwise the probability of leaving the core would be $1 > \epsilon_M$. Moreover, a random infinite path in $\bar{M}_{G,k}$ reaches t with probability 1 so as $\epsilon_M < 1$ the state t must also be in the core. Hence, there are at most $k_{G,k} - 2 = 3 \cdot n^2 + k$ remaining states in the core. Now, by Observation 3, we may remove states of chains that are not fully contained in the core to obtain an $(\epsilon_M, k_{G,k})$ -core whose remainder consists only of whole chains. Since chains have length $3 \cdot n$ or $3 \cdot n + 1$ and $3 \cdot n \cdot (n + 1) > 3 \cdot n^2 + k$, the core must contain states of at most n chains. On the other hand, a random infinite path in $\bar{M}_{G,k}$ leaves the core with probability at least p_1 for every chain not fully contained in the core. So as $(n + 1) \cdot p_1 > n \cdot p_1 + (n - k) \cdot (n - k - 1) \cdot p_1 \cdot p_2 = \epsilon_M$ due to $p_2 = \frac{1}{n^{10}}$, we may conclude that the core must contain all states from at least n chains as otherwise the probability of leaving the core via an edge of probability p_1 would be at least $(n + 1) \cdot p_1$. Thus, the core must consist of $v_{G,k}$, t and all states in exactly n chains. Finally, since the core is of size at most $k_{G,k} = 3 \cdot n^2 + k + 2$ and it must contain $v_{G,k}$ and t , we conclude that at most k of these chains are from the third layer. This proves Observation 4.

► **Observation 5.** *If $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core, then it also contains an $(\epsilon_M, k_{G,k})$ -core such that, for each $1 \leq i \leq n$, the core contains either all states from the i -th chain in the second layer or all states from the i -th chain in the third layer.*

Let C be a set of states in $\bar{M}_{G,k}$ that contains $v_{G,k}$, t and all states of exactly n chains where at most k chains are in the third layer. Observation 4 implies that at least one core of such form exists whenever $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core. We show that, if C did not satisfy the property in Observation 5, then the probability of leaving C would strictly exceed ϵ_M and thus it would not be an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$. First, by our constructions of $M_{G,k}$ and $\bar{M}_{G,k}$, for each chain in $\bar{M}_{G,k}$ the probability of an edge from the initial state $v_{G,k}$ to the first state in the chain is equal to p_1 . Hence, since we assume that C contains exactly n chains and does not contain any states in the remaining n chains, the probability of leaving the core in exactly one step is $n \cdot p_1$. On the other hand, for each $1 \leq i \leq n$, the probability of first entering the i -th chain in the second layer and then moving to a chain in the third layer that is not contained in C is at least:

- $(n - k) \cdot p_1 \cdot p_2$, if the i -th chain in the third layer is also contained in C ;
- $(n - k - 1) \cdot p_1 \cdot p_2$, otherwise.

This is because C contains at most k chains in the third layer, and the last state in the i -th chain in the second layer is connected to the first states of all but the i -th chains in the third layer by an edge of probability p_2 . Therefore, as C contains at least $n - k$ chains in the second layer, we conclude that the probability of leaving C is at least $n \cdot p_1 + (n - k) \cdot (n - k - 1) \cdot p_1 \cdot p_2 = \epsilon_M$, with the equality attained if and only if there is no $1 \leq i \leq n$ for which both the i -th chain in the second layer and the i -th chain in the third layer are contained in C . Due to the assumption that C contains all states of exactly n chains, it follows that for each $1 \leq i \leq n$ the set C should contain either the i -th chain in the second layer or the i -th chain in the third layer for the probability of leaving C not to exceed ϵ_M . This concludes the proof of Observation 5.

► **Observation 6.** *If $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core, then it also contains an $(\epsilon_M, k_{G,k})$ -core C for which there does not exist an edge in $\bar{M}_{G,k}$ of probability p_3 whose source state is in C but target state is not in C .*

Suppose that $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core. Let C be an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$ which satisfies the properties in Observation 5. It follows from the proof of Observation 5 that the probability of a random infinite path in $\bar{M}_{G,k}$ leaving C must be exactly ϵ_M and that this probability is attained solely by taking edges of probabilities p_1 and p_2 . Hence, there may not exist an edge in $\bar{M}_{G,k}$ of probability p_3 whose source state is in C but target state is not in C , which proves Observation 6.

► **Observation 7.** *If $\bar{M}_{G,k}$ contains an $(\epsilon_M, k_{G,k})$ -core then G has a vertex cover of size k . Hence, from Observation 1 we may conclude that $(M_{G,k}, k_{G,k}) \in \text{CORE}_\epsilon$ implies $(G, k) \in \text{VERTEX-COVER}$.*

Let C be an $(\epsilon_M, k_{G,k})$ -core in $\bar{M}_{G,k}$ that satisfies the properties in the previous observations. We use it to construct a vertex cover of size at most k in G and therefore prove Observation 7. Observations 4 and 5 imply that C must contain $v_{G,k}$, t and all states of exactly n chains of which at most k are in the third layer, such that for each $1 \leq i \leq n$, C contains either the i -th chain in the second layer or the i -th chain in the third layer. Furthermore, by Observation 6 there does not exist an edge in $\bar{M}_{G,k}$ of probability p_3 whose source state is in C but target state is not in C . But, by our construction of $M_{G,k}$ and $\bar{M}_{G,k}$, recall that we have an edge of probability p_3 from the last state in the i -th chain in the second layer to the first state of the j -th chain in the third layer if and only if $i < j$ and (v_i, v_j) is an edge in G . For this not to be an edge whose source state is contained in C but target state is not contained in C , we must either have that the i -th chain in the second layer is not contained in C and therefore the i -th chain in the third layer is contained in C , or that the i -th chain in the second layer and the j -th chain in the third layer are both contained in C . Thus, for each edge (v_i, v_j) in G with $i < j$, the core C should contain at least one of the i -th chain in the third layer or the j -th chain in the third layer. Therefore, as a core must contain at most k chains in the third layer, defining

$$K = \{i \in \{1, \dots, n\} \mid C \text{ contains the } i\text{-th chain in the third layer}\}.$$

gives rise to a vertex cover of size at most k in G . This concludes the proof. ◀

29:10 Algorithms and Hardness Results for Computing Cores of Markov Chains

Theorem 1 shows that the problem of deciding whether a Markov chain contains an ϵ -core of at most the given size k is NP-complete. Furthermore, Markov chains $M_{G,k}$ constructed in the proof of Theorem 1 are *acyclic*. This is because edges in $M_{G,k}$ only connect states from lower indexed layers to states in upper indexed layers, and the only cycle contained within a layer is a probability 1 self-loop at the state t in the fourth layer.

In the following theorem, we show that the CORE_ϵ problem remains NP-complete even if we restrict it to acyclic Markov chains that furthermore have maximal vertex degree of at most 3. Hence, even parametrizing Markov chains by the maximal vertex degree would not make the problem solvable in polynomial time, which indicates that the problem of computing cores of at most a given size is computationally a very challenging problem. While Theorem 8 generalizes the result of Theorem 1, the reason why we present them separately is that the construction in the proof of Theorem 8 is more complicated than that in Theorem 1. To that end, we only note that the proof of Theorem 8 modifies the construction of Theorem 1 in a way which bounds the maximal vertex degree of $M_{G,k}$, and we defer the details of this modification to Appendix A.

► **Theorem 8** (Proof in Appendix A). *For an $\epsilon \in (0, 1)$, let CORE_ϵ^* be the language defined as*

$$\left\{ (M, k) \mid M \text{ is an acyclic Markov chain of maximal} \right. \\ \left. \text{vertex degree } \leq 3 \text{ that contains an } (\epsilon, k)\text{-core} \right\}.$$

Then, CORE_ϵ^ is NP-complete for any $\epsilon \in (0, 1)$.*

4 A PTIME Algorithm for Optimal Cores in MCs over CFGs

In this section, we provide a PTIME algorithm that, given a threshold $\epsilon \in (0, 1)$ and a Markov chain M over the control-flow graph of a program P , outputs an optimal ϵ -core of M , i.e. an ϵ -core of minimum possible size. As mentioned in Section 1, MCs over CFGs are naturally obtained whenever (statistical) branch prediction is applied to a program. As such, finding a core in such MCs can directly help find a set of lines in the program that covers the vast majority of the runs and is left with very low probability. Any desired static analysis can then be limited to the lines in the core, leading to rigorous probabilistic bounds for the desired property. MCs over CFGs have also been studied in [1].

Before presenting our algorithm, in Section 4.1 we first formally present a grammar for an imperative probabilistic programming language that can define both P and M at the same time. We then present our PTIME algorithm in Section 4.2.

4.1 Markov Chains Induced by Probabilistic Programs

Syntax Grammar. We consider a fragment of finite state first-order imperative probabilistic programs defined by the following grammar:

$$\begin{aligned} \langle prog \rangle = & \text{atomic} \\ & | \text{if } \mathbf{prob}(p) \langle prog \rangle \text{ else } \langle prog \rangle \\ & | \text{while } \mathbf{prob}(p) \text{ do } \langle prog \rangle \\ & | \langle prog \rangle ; \langle prog \rangle \end{aligned} \tag{1}$$

In both the second and the third case, $p \in (0, 1)$ is a probability parameter. The first case considers trivial programs that execute a single statement and immediately terminate. The second case considers probabilistic if-branching, where the control-flow follows the if-branch and executes the first program with probability p , and it follows the else-branch and executes the second program with probability $1 - p$. The third case considers a construct for probabilistic loops, where, in each iteration, the control-flow enters the loop and executes the inner-nested program with probability p upon which it returns to the loop entry, and it leaves the loop and terminates with probability $1 - p$. Finally, the fourth case considers sequential composition of two programs, where the second program is executed upon termination of the first program.

In Markov chains induced by probabilistic programs, in addition to the initial state we also assume the existence of a designated *terminal state*. Thus, for the rest of this section, we slightly modify our definition of Markov chains in Section 2 and define them as tuples $M = (V, s, \delta, t)$, where the last element $t \in V$ denotes the terminal state.

MCs induced by Probabilistic Programs. We now formally define how a probabilistic program generated by the above syntax grammar induces a Markov chain. Given a probabilistic program P , consider its parse tree according to the grammar. In order to define the Markov chain M_P that is induced by P , we start by constructing a Markov chain associated to each leaf node in the parse tree and traverse the parse tree bottom-up in order to construct Markov chains associated to parent programs. The Markov chain M_P associated to the probabilistic program P is then defined to be the Markov chain associated to the root program in the parse tree.

MCs for Leaves. The leaves of the parse tree are trivial **atomic** statements. A Markov chain associated to an **atomic** statement is defined via $M_{\text{atomic}} = (V_{\text{atomic}}, s, \delta_{\text{atomic}}, t)$, where

- The state space consists of two states $V_{\text{atomic}} = \{s, t\}$, and
- the probabilistic transition function is defined via $\delta_{\text{atomic}}(s)(t) = \delta_{\text{atomic}}(t)(t) = 1$.

Non-leaf nodes in the parse tree correspond to subprograms that are obtained either by probabilistic if-branching, probabilistic loops or sequential composition constructs. We consider each of the three cases and describe how a Markov chain associated to a parent node program is constructed from Markov chains associated to the children node programs.

Branching Nodes. Suppose that a parent node program is given by

if **prob**(p) $prog_1$ else $prog_2$,

and let $M_1 = (V_1, s_1, \delta_1, t_1)$ and $M_2 = (V_2, s_2, \delta_2, t_2)$ be the Markov chains associated to $prog_1$ and $prog_2$, respectively. To construct the Markov chain associated to the parent node program, we introduce two new states s and t and consider $M = (V, s, \delta, t)$, where $V = \{s, t\} \cup V_1 \cup V_2$ and $\delta : V \rightarrow \mathcal{D}(V)$ is defined via

- $\delta(s)(s_1) = p, \delta(s)(s_2) = 1 - p,$
- $\delta(v) = \delta_1(v)$ for each $v \in V_1 \setminus \{t_1\},$
- $\delta(v) = \delta_2(v)$ for each $v \in V_2 \setminus \{t_2\},$
- $\delta(t_1)(t) = \delta(t_2)(t) = 1,$ and
- $\delta(t)(t) = 1.$

29:12 Algorithms and Hardness Results for Computing Cores of Markov Chains

Intuitively, the Markov chain M has initial state s from which a random infinite path either moves to the initial state s_1 of M_1 with probability p , or to the initial state s_2 of M_2 with probability $1 - p$. Then, upon reaching terminal states t_1 of M_1 or t_2 of M_2 , a random infinite path moves to the terminal state t of M and stays there indefinitely due to the probability 1 self-loop.

Loop Nodes. Suppose that a parent node program is given by

while **prob**(p) **do** $prog_1$,

and let $M_1 = (V_1, s_1, \delta_1, t_1)$ be the Markov chains associated to $prog_1$. To construct the Markov chain associated to the parent node program, we introduce two new states s and t and consider $M = (V, s, \delta, t)$, where $V = \{s, t\} \cup V_1$ and $\delta : V \rightarrow \mathcal{D}(V)$ is defined via

- $\delta(s)(s_1) = p, \delta(s)(t) = 1 - p,$
- $\delta(v) = \delta_1(v)$ for each $v \in V_1 \setminus \{t_1\},$
- $\delta(t_1)(s) = 1,$ and
- $\delta(t)(t) = 1.$

Intuitively, the Markov chain M has initial state s from which a random infinite path either moves to the initial state s_1 of M_1 with probability p , or to the terminal state t of M with probability $1 - p$ where it stays indefinitely due to the probability 1 self-loop. Then, upon reaching terminal states t_1 of M_1 , a random infinite path moves to the initial state s of M with probability 1.

Sequential Composition Nodes. Finally, suppose that a parent node program is

$prog_1 ; prog_2,$

and let $M_1 = (V_1, s_1, \delta_1, t_1)$ and $M_2 = (V_2, s_2, \delta_2, t_2)$ be the Markov chains associated to $prog_1$ and $prog_2$, respectively. To construct the Markov chain associated to the parent node program, we introduce two new states s and t and consider $M = (V, s, \delta, t)$, where $V = \{s, t\} \cup V_1 \cup V_2$ and $\delta : V \rightarrow \mathcal{D}(V)$ is defined via

- $\delta(s)(s_1) = 1, \delta(t_1)(s_2) = 1, \delta(t_2)(t) = 1,$
- $\delta(v) = \delta_1(v)$ for each $v \in V_1 \setminus \{t_1\},$
- $\delta(v) = \delta_2(v)$ for each $v \in V_2 \setminus \{t_2\},$ and
- $\delta(t)(t) = 1.$

Intuitively, the Markov chain M has initial state s from which a random infinite path with probability 1 moves to the initial state s_1 of M_1 . Then, upon reaching the terminal state t_1 of M_1 , it with probability 1 moves to the initial state s_2 of M_2 . Finally, upon reaching the terminal state t_2 of M_2 , it with probability 1 moves to the terminal state t of M where it stays indefinitely due to the probability 1 self-loop.

It is easy to see that the Markov chains obtained above are over the CFG of their corresponding program and that, conversely, any MC over a CFG can be obtained by a probabilistic program generated by our grammar.

4.2 PTIME Algorithm for Optimal Core Computation

We now present our polynomial time algorithm for computing the size of a smallest ϵ -core in a Markov chain induced by a probabilistic program generated by the syntax grammar in Equation (1). Given $\epsilon \in [0, 1]$ and a probabilistic program P that can be generated by the grammar in Equation (1), our algorithm first constructs its parse tree T_P . It then performs dynamic programming on the parse tree. In particular, it traverses the parse tree bottom-up and for each subprogram P' with associated Markov chain $M_{P'}$ it computes a sequence of non-negative real numbers $(a_{P'}[0], a_{P'}[1], \dots, a_{P'}[|V_{P'}|])$, where $|V_{P'}|$ is the number of states in $M_{P'}$ and

$$a_{P'}[k] = \min \left\{ \epsilon' \geq 0 \mid M_{P'} \text{ contains an } (\epsilon', k)\text{-core} \right\}$$

for each $0 \leq k \leq |V_{P'}|$. In other words, the k -th entry in the sequence is the minimal probability threshold $\epsilon' \geq 0$ with which a set of k states in the Markov chain $M_{P'}$ may be left. Then, once it has computed the sequence $(a_P[0], \dots, a_P[|V_P|])$ for the root node program P , the algorithm can immediately conclude that the minimal size of an ϵ -core in P is

$$k_{\min} = \min \left\{ 1 \leq k \leq |V_P| \mid a_P[k] \leq \epsilon \right\}.$$

In what follows, we describe how our algorithm computes such a sequence for each program in the parse tree T_P of P . We then prove the correctness of our algorithm and that it runs in polynomial time.

Dynamic Programming on the Parse Tree. The algorithm starts by computing the sequence $(a_{P'}[0], a_{P'}[1], \dots, a_{P'}[|V_{P'}|])$ from each leaf node program P' in the parse tree, upon which it traverses the parse tree bottom-up in order to compute the sequence for each node in the tree. We now describe the dynamic programming steps for each construct type in the syntax grammar:

- *Atomic statement at a leaf node.* Recall that the Markov chain associated to an atomic statement is defined via $M_{\text{atomic}} = (V_{\text{atomic}}, s, \delta_{\text{atomic}}, t)$ with $V_{\text{atomic}} = \{s, t\}$ and $\delta_{\text{atomic}}(s)(t) = \delta_{\text{atomic}}(t)(s) = 1$. Hence, its state space consists of 2 states and its sequence $(a_{\text{atomic}}[0], a_{\text{atomic}}[1], a_{\text{atomic}}[2])$ is defined via

$$a_{\text{atomic}}[k] = \begin{cases} 1, & k = 0, 1 \\ 0, & k = 2 \end{cases}$$

One can easily verify by inspection that each $a_{\text{atomic}}[k]$ is indeed the minimal probability with which a set of k states in M_{atomic} may be left.

- *Probabilistic if-branching node.* Consider now the parse tree node that corresponds to the probabilistic if-branching

$$\text{prog} = \text{if } \mathbf{prob}(p) \text{ } \text{prog}_1 \text{ else } \text{prog}_2,$$

and let $(a_{\text{prog}_1}[0], \dots, a_{\text{prog}_1}[|V_1|])$ and $(a_{\text{prog}_2}[0], \dots, a_{\text{prog}_2}[|V_2|])$ be the sequences that the algorithm has computed for the child node programs prog_1 and prog_2 in the parse tree T_P . The algorithm then sets

$$a_{\text{prog}}[k] = \begin{cases} 1, & k = 0, 1 \\ \min_{j_1, j_2 \geq 0, j_1 + j_2 = k-2} \left(p \cdot a_{\text{prog}_1}[j_1] + (1-p) \cdot a_{\text{prog}_2}[j_2] \right), & k \geq 2 \end{cases}$$

29:14 Algorithms and Hardness Results for Computing Cores of Markov Chains

To see that this formula indeed defines the minimal probability with which a set of k states in the Markov chain M_{prog} of $prog$ may be left, note that for any positive probability threshold $\epsilon' > 0$, an ϵ' -core in M_{prog} must contain the source state, the terminal state and the total of $k - 2$ states in the Markov chains M_{prog_1} of $prog_1$ and M_{prog_2} of $prog_2$. Thus, if we denote by j_1 and j_2 the number of these $k - 2$ states that are contained in M_{prog_1} and M_{prog_2} , by the correctness of the algorithm for child node programs we conclude that the above formula minimizes the probability with which a set of k states in the Markov chain M_{prog} of $prog$ may be left.

- *Probabilistic loop node.* Next, consider the parse tree node that corresponds to the probabilistic loop

$$prog = \text{while } \mathbf{prob}(p) \text{ do } prog_1,$$

and let $(a_{prog_1}[0], \dots, a_{prog_1}[|V_1|])$ be the sequence that the algorithm has computed for the child node program $prog_1$ in the parse tree T_P . The algorithm sets

$$a_{prog}[k] = \begin{cases} 1, & k = 0, 1 \\ \frac{p \cdot a_{prog_1}[k-2]}{1 - p \cdot (1 - a_{prog_1}[k-2])}, & k \geq 2 \end{cases}$$

To see that this formula defines the minimal probability with which a set of k states in the Markov chain M_{prog} of $prog$ may be left, note that for any positive probability threshold $\epsilon' > 0$, an ϵ' -core in M_{prog} must contain the source state, the terminal state and the total of $k - 2$ states in the Markov chain M_{prog_1} of $prog_1$. The algorithm starts each new loop iteration with probability p , and by the correctness of the algorithm for the root node program we know that in each loop iteration it leaves the set of $k - 2$ states in M_{prog_1} with probability at most $a_{prog_1}[k - 2]$. Hence, by summing up the geometric series, we conclude that a set of k states in the Markov chain M_{prog} of $prog$ may be left with probability at most

$$p \cdot a_{prog_1}[k - 2] \cdot \sum_{i=0}^{\infty} \left(p \cdot (1 - a_{prog_1}[k - 2]) \right)^i = \frac{p \cdot a_{prog_1}[k - 2]}{1 - p \cdot (1 - a_{prog_1}[k - 2])}.$$

- *Sequential decomposition node.* Finally, consider the parse tree node that corresponds to the sequential composition

$$prog = prog_1 ; prog_2,$$

and let $(a_{prog_1}[0], \dots, a_{prog_1}[|V_1|])$ and $(a_{prog_2}[0], \dots, a_{prog_2}[|V_2|])$ be the sequences that the algorithm has computed for the child node programs $prog_1$ and $prog_2$ in the parse tree T_P . The algorithm sets

$$a_{prog}[k] = \begin{cases} 1, & k = 0, 1 \\ \min_{j_1, j_2 \geq 0, j_1 + j_2 = k - 2} \left(a_{prog_1}[j_1] + (1 - a_{prog_1}[j_1]) \cdot a_{prog_2}[j_2] \right), & k \geq 2 \end{cases}$$

To see that this formula indeed defines the minimal probability with which a set of k states in the Markov chain M_{prog} of $prog$ may be left, note that for any positive probability threshold $\epsilon' > 0$, an ϵ' -core in M_{prog} must contain the source state, the terminal state and the total of $k - 2$ states in the Markov chains M_{prog_1} of $prog_1$ and M_{prog_2} of $prog_2$. Thus,

if we denote by j_1 and j_2 the number of these $k - 2$ states that are contained in M_{prog_1} and M_{prog_2} , by the correctness of the algorithm for child node programs we conclude that the above formula minimizes the probability with which a set of k states in the Markov chain M_{prog} of $prog$ may be left.

Analysis of each case above and induction on the depth of the parse tree T_P allows us to conclude the following theorem.

► **Theorem 9.** *Let P be a probabilistic program generated by the syntax grammar in Equation (1) and let M_P be the Markov chain induced by P with state space V_P . Let $(a_P[0], \dots, a_P[|V_P|])$ be the sequence that the algorithm computes for the program P . Then, for each $0 \leq k \leq |V_P|$, it holds that*

$$a_P[k] = \min \left\{ \epsilon' \geq 0 \mid M_P \text{ contains an } (\epsilon', k)\text{-core} \right\}.$$

Note that this immediately shows the correctness of our algorithm.

Runtime Analysis. Let $n = |M_P|$ be the size of the Markov chain induced by a probabilistic program P . Note that our dynamic programming algorithm processes each node in the parse tree T_P in $\mathcal{O}(n^2)$ time. Hence, as the size of the parse tree is linear in n , we conclude that the algorithm runs in $\mathcal{O}(n^3)$ time and therefore has polynomial runtime in the size of the underlying Markov chain. Moreover, note that the n in turn is linear in the size of the parse tree which is linear in the size of the program code.

5 Conclusion

An ϵ -core in a Markov chain is a set of states that is left by a random run with probability at most ϵ . In this work, we considered the problem of finding an optimal (smallest) ϵ -core in a given Markov chain M . For every $\epsilon \notin \{0, 1\}$, we proved that the problem is NP-hard. Moreover, this NP-hardness is preserved even when we limit our instances to acyclic MCs with bounded degree 3. Our NP-hardness result answered an open problem posed by [26]. We then showed that for Markov chains over control-flow graphs of structured programs, the problem can be solved in PTIME. In summary, our results demonstrate that there is no efficient algorithm for the general case of the problem unless $P=NP$, and hence practitioners should use sampling and partial-exploration algorithms with no theoretical guarantees of optimality, such as the one provided by [26]. However, in the important special case of MCs over CFGs, a PTIME algorithm exists.

References

- 1 Ali Asadi, Krishnendu Chatterjee, Amir Kafshdar Goharshady, Kiarash Mohammadi, and Andreas Pavlogiannis. Faster algorithms for quantitative analysis of MCs and MDPs with small treewidth. In *ATVA*, pages 253–270, 2020.
- 2 Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Kretínský. Continuous-time Markov decisions based on partial exploration. In *ATVA*, pages 317–334, 2018.
- 3 Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggen-dorfer. Value iteration for long-run average reward in Markov decision processes. In *CAV*, pages 201–221, 2017.
- 4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

- 5 Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Relatively complete verification of probabilistic programs: an expressive language for expectation-based reasoning. In *POPL*, 2021.
- 6 Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, pages 98–114, 2014.
- 7 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through positivstellensatz’s. In *CAV*, pages 3–22, 2016.
- 8 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Non-polynomial worst-case analysis of recursive programs. In *CAV*, pages 41–63, 2017.
- 9 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Non-polynomial worst-case analysis of recursive programs. *ACM Trans. Program. Lang. Syst.*, 41(4):20:1–20:52, 2019.
- 10 Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Nastaran Okati. Computational approaches for stochastic shortest path on succinct MDPs. In *IJCAI*, pages 4700–4707, 2018.
- 11 Krishnendu Chatterjee, Amir Kafshdar Goharshady, Rasmus Ibsen-Jensen, and Yaron Velner. Ergodic mean-payoff games for the analysis of attacks in crypto-currencies. In *CONCUR*, pages 11:1–11:17, 2018.
- 12 Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Đorđe Žikelić. Sound and complete certificates for quantitative termination analysis of probabilistic programs. In *CAV*, 2022.
- 13 Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. Probabilistic smart contracts: Secure randomness on the blockchain. In *ICBC*, pages 403–412, 2019.
- 14 Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Yaron Velner. Quantitative analysis of smart contracts. In *ESOP*, volume 10801, pages 739–767, 2018.
- 15 Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, Jiri Závěručky, and Đorđe Žikelić. On lexicographic proof rules for probabilistic termination. In *FM*, volume 13047 of *Lecture Notes in Computer Science*, pages 619–639. Springer, 2021.
- 16 Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic invariants for probabilistic termination. In *POPL*, pages 145–160, 2017.
- 17 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, pages 592–600, 2017.
- 18 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.
- 19 Paul A Gagnic. *Markov chains: from theory to implementation and experimentation*. Wiley, 2017.
- 20 Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4):403–434, 1976.
- 21 S. Gómez, A. Arenas, J. Borge-Holthoefer, S. Meloni, and Y. Moreno. Discrete-time Markov chain approach to contact-based disease spreading in complex networks. *EPL*, 89(3), 2010.
- 22 Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Modular verification for almost-sure termination of probabilistic programs. In *OOPSLA*, pages 129:1–129:29, 2019.
- 23 Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.
- 24 Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Informatica*, 56(3):255–285, 2019.

- 25 Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.
- 26 Jan Kretínský and Tobias Meggendorfer. Of cores: A partial-exploration framework for Markov decision processes. *Log. Methods Comput. Sci.*, 16(4), 2020.
- 27 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.
- 28 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.
- 29 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. CRC Press, 1999.
- 30 Johan Paulsson. Summing up the noise in gene networks. *Nature*, 427(6973):415–418, 2004.
- 31 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.
- 32 Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical society*, 74(2):358–366, 1953.
- 33 James E. Smith. A study of branch prediction strategies. In *ISCA*, pages 202–215, 1998.
- 34 Jana Wagemaker, Nate Foster, Tobias Kappé, Dexter Kozen, Jurriaan Rot, and Alexandra Silva. Concurrent NetKAT – modeling and analyzing stateful, concurrent networks. In *ESOP*, pages 575–602, 2022.
- 35 Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Quantitative analysis of assertion violations in probabilistic programs. In *PLDI*, pages 1171–1186. ACM, 2021.
- 36 Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. Cost analysis of nondeterministic probabilistic programs. In *PLDI*, pages 204–220. ACM, 2019.

A Proof of Theorem 8

In Theorem 1, we showed that CORE_ϵ is contained in NP. Hence, as acyclic and the maximal vertex degree conditions may be checked in linear time, we conclude that CORE_ϵ^* is also in NP.

To prove that CORE_ϵ^* is NP-hard, we again show a reduction from the VERTEX-COVER problem. Fix $\epsilon \in (0, 1)$ and consider an instance $(G = (V, E), k)$ of the VERTEX-COVER problem where $n = |V| > \frac{1}{\epsilon}$, $n > 20$ and $n > \frac{1}{\sqrt{1-\epsilon}}$ (these assumptions are required as some inequalities below will only hold for sufficiently large values of n). The reduction is obtained by modifying our construction of $(M_{G,k}, k_{G,k})$ in the proof of Theorem 1 in a way that makes $M_{G,k}$ have the maximal vertex degree of 3 while keeping it acyclic. We denote by $(M_{G,k}^*, k_{G,k}^*)$ the newly constructed Markov chain. In what follows, we outline the differences in the construction:

- *Core size.* We set $k_{G,k}^* = 4n^2 + 4n + k$, which is polynomial in the sizes of G and k .
- *Splitting of the initial state.* The initial state $v_{G,k}$ in $M_{G,k}$ is replaced by $2n$ states $v_1^{1,*}, \dots, v_{2n}^{1,*}$, where the first state $v_1^{1,*}$ in the ordering is designated as the initial state of $M_{G,k}^*$. This is done in order to ensure that the maximal vertex degree of each state in the first layer of the Markov chain $M_{G,k}^*$ is at most 3. Note, $v_1^{1,*}, \dots, v_{2n}^{1,*}$ do not form a chain as the probabilities of transitions between successive states will not be 1. Then, for each $1 \leq i \leq n$, we set $\delta_{G,k}(v_i^{1,*})(v_{i,1}^2) = p_{1,i}$ and $\delta_{G,k}(v_{n+i}^{1,*})(v_{i,1}^3) = p_{1,n+i}$, where $p_{1,i}$ and $p_{1,n+i}$ are probabilities that are defined inductively as follows. For $i = 1$, we set $p_{1,1} = p_1 = \frac{1}{n^3}$. For $2 \leq i \leq 2n$, we set

$$p_{1,i} = \frac{p_1}{\prod_{j=1}^{i-1} (1 - p_{1,j})}.$$

Then, connect the last state $v_{2n}^{1,*}$ to the first state v_1^1 of the chain $v_1^1, \dots, v_{n^3}^1$ of length n^3 by an edge with probability

$$\delta_{G,k}(v_{2n}^{1,*}, v_1^1) = \frac{\epsilon - \epsilon_M}{\prod_{j=1}^{2n-1} (1 - p_{1,j})}$$

so that the probability of reaching the chain of length n^3 from the initial state $v_1^{1,*}$ is equal to $\epsilon - \epsilon_M$ as in the proof of Theorem 1.

Finally, for each $1 \leq i < 2n$, we set $\delta_{G,k}(v_i^{1,*})(v_{i+1}^{1,*}) = 1 - p_{1,i}$ and we set $\delta_{G,k}(v_{2n}^{1,*})(t_1^*) = 1 - p_{1,2n} - \frac{\epsilon - \epsilon_M}{\prod_{j=1}^{2n-1} (1 - p_{1,j})}$, where t_1^* is the first state in the fourth layer that will be defined in what follows.

To prove that each $p_{1,i} \in [0, 1]$, we show by induction on i that $p_{1,i} < \frac{1}{n^2}$. Indeed, $p_{1,1} = \frac{1}{n^3} < \frac{1}{n^2}$ and for $i > 1$ by induction hypothesis we have

$$\begin{aligned} p_{1,i} &= \frac{p_1}{\prod_{j=1}^{i-1} (1 - p_{1,j})} \leq \frac{p_1}{(1 - 1/n^2)^{2n}} \leq \frac{p_1}{(1 - 1/n)^{2n}} \\ &= \frac{1/n^3}{(1 - 1/n)^{2n-2} (1 - 1/n)^2} < \frac{e^2}{n(n-1)^2} < \frac{1}{n^2}, \end{aligned}$$

for $n \geq 20$, where we use that $(1 - 1/x)^{-x+1} < e$ for $x > 0$.

The choices of probabilities ensure that a random infinite path in $M_{G,k}^*$ is equally likely to traverse an edge from a state in the first layer to the first state of each chain in the second or the third layer and that this happens with probability p_1 for each chain. Furthermore, this construction ensures that each state in the first layer has a vertex degree of at most 3. Finally, due to the assumption that $n > \frac{1}{\sqrt{1-\epsilon}}$ which is equivalent to $\epsilon < 1 - \frac{1}{n^2}$, we have that the probability of reaching $v_{2n}^{1,*}$ from the initial state $v_1^{1,*}$ is at least

$$\prod_{i=1}^{2n-1} (1 - p_{1,i}) > (1 - \frac{1}{n^2})^{2n-1} \geq 1 - \frac{1}{n^2} > \epsilon,$$

therefore an ϵ -core in $M_{G,k}$ must contain all states in the sequence $v_1^{1,*}, \dots, v_{2n}^{1,*}$.

- *Extension of sequence lengths.* Sequences in the second layer of $M_{G,k}^*$ have length $4n$, instead of length $3n$ as was the case in our construction of $M_{G,k}$. As in the proof of Theorem 1, for each $1 \leq i \leq n$ and $1 \leq j \leq 4n$, we use $v_{i,j}^2$ to denote the j -th state along the i -th sequence. We note that in $M_{G,k}^*$ these sequences *do not* form chains as the probabilities of transitions between successive states will not be 1, as we specify below. Similarly, sequences in the third layer of $M_{G,k}^*$ have length $4n + 1$ instead of length $3n + 1$ as was the case in $M_{G,k}$ and again they *do not* form chains. For each $1 \leq i \leq n$ and $1 \leq j \leq 4n + 1$, we use $v_{i,j}^3$ to denote the j -th state along the i -th sequence.
- *Redistributing transition probabilities p_2 in order to bound degrees.* Recall, for each $1 \leq i \leq n$ and each $j \neq i$, in $M_{G,k}$ we had an edge from $v_{i,3n}^2$ to $v_{j,1}^3$ with $\delta_{G,k}(v_{i,3n}^2)(v_{j,1}^3) = p_2$. We now remove each such edge, and replace it with a new edge from $v_{i,j}^2$ to $v_{j,i}^3$ which has probability $p_{2,i,j}$. The new probabilities are defined as follows. For $i = j$, we set $p_{2,i,i} = 0$. Otherwise, we set $p_{2,i,1} = p_2 = \frac{1}{n^{10}}$ for each $i > 1$ and then for $j \neq i$ and $j > 1$ we inductively define

$$p_{2,i,j} = \frac{p_2}{\prod_{l=1}^{j-1} (1 - p_{2,i,l})}.$$

For each $1 \leq i \leq n$, we prove that each $p_{2,i,j} \in [0, 1]$ by showing that each $p_{2,i,j} < \frac{1}{n^9}$ by induction on j . Indeed, $p_{2,i,1} = 0$ if $i = 1$ and $p_{2,i,1} = \frac{1}{n^{10}} < \frac{1}{n^9}$ if $i > 1$, as $n \geq 20$. Then, for $j > 1$, by induction hypothesis we have

$$\begin{aligned} p_{2,i,j} &= \frac{p_2}{\prod_{l=1}^{j-1} (1 - p_{2,i,l})} \leq \frac{p_2}{(1 - 1/n)^n} = \frac{1/n^{10}}{(1 - 1/n)^n} \\ &< \frac{1/n^{10}}{(1 - 1/n)^{n-1} (1 - 1/n)} < \frac{e}{n^9(n-1)} < \frac{1}{n^9}. \end{aligned}$$

Finally, for each $v_{i,j}^3$ with $j \leq n$, we set $\delta_{G,k}(v_{i,j}^3)(v_{i,j+1}^3) = 1 - p_{2,i,j}$.

This construction ensures that, once a random infinite path in $M_{G,k}$ reaches the first state $v_{i,1}^2$ of the i -th chain in the second layer, it is then equally likely to traverse the sequence $v_{i,1}^2, \dots, v_{i,4n}^2$ up to the j -th state $v_{i,j}^2$ and then to move to the state $v_{j,i}^3$ of the j -th chain in the third layer and that this happens with probability p_2 for each $1 \leq j \leq n$ with $j \neq i$. Furthermore, it ensures that the vertex degrees due to edges of probability p_2 from the second to the third layer do not exceed 3.

- *Redistributing transition probabilities p_3 in order to bound degrees.* Recall, for each edge (v_i, v_j) in G with $i < j$, in $M_{G,k}$ we had an edge from $v_{i,3n}^2$ to $v_{j,1}^3$ of probability $\delta_{G,k}(v_{i,3n}^2)(v_{j,1}^3) = p_3$ where $p_3 = \frac{1}{n^{50}}$.

We now remove this edge and replace them with a new edge from $v_{i,n+j}^2$ to $v_{j,n+i}^3$ of probability $\delta_{G,k}(v_{i,n+j}^2)(v_{j,n+i}^3) = p_{3,i,j}$. These probabilities are defined inductively via

$$p_{3,i,j} = \frac{p_3}{\prod_{l=1}^n (1 - p_{2,i,l}) \prod_{l=1}^{j-1} (1 - p_{3,i,l})}$$

for each $i < j$ for which (v_i, v_j) is an edge in G , with $p_{3,i,j} = 0$ whenever $i \geq j$ or when (v_i, v_j) is not an edge in G .

For each $1 \leq i \leq n$, we prove that each $p_{3,i,j} \in [0, 1]$ by showing that $p_{3,i,j} < \frac{1}{n^{49}}$ by induction on j . Indeed, $p_{3,i,1} = 0$ if (v_i, v_j) is not an edge in G and $p_{3,i,1} = p_3 = \frac{1}{n^{50}} < \frac{1}{n^{49}}$ if (v_i, v_j) is an edge in G as $n \geq 20$. Then, for $j > 1$, by induction hypothesis we have

$$\begin{aligned} p_{3,i,j} &= \frac{p_3}{\prod_{l=1}^n (1 - p_{2,i,l}) \prod_{l=1}^{j-1} (1 - p_{3,i,l})} \leq \frac{1/n^{50}}{(1 - 1/n)^{2n}} \\ &= \frac{1/n^{50}}{(1 - 1/n)^{2n-2} (1 - 1/n)^2} < \frac{e^2}{n^{48}(n-1)^2} < \frac{1}{n^{49}}. \end{aligned}$$

Together with the splitting of edges of probability p_2 that we constructed above, this construction ensures that, once a random infinite path in $M_{G,k}$ reaches the first state $v_{i,1}^2$ of the i -th chain in the second layer, it either

- traverses the sequence $v_{i,1}^2, \dots, v_{i,4n}^2$ up to the j -th state $v_{i,j}^2$ and then to move to the state $v_{j,i}^3$ of the j -th chain in the third layer with probability p_2 for each $j \neq i$, or
- it traverses the sequence $v_{i,1}^2, \dots, v_{i,4n}^2$ up to the $(n+j)$ -th state $v_{i,n+j}^2$ and then to move to the state $v_{j,n+i}^3$ of the j -th chain in the third layer with probability p_3 for each $i < j$ such that (v_i, v_j) is an edge in G , or

29:20 Algorithms and Hardness Results for Computing Cores of Markov Chains

- it traverses the sequence $v_{i,1}^2, \dots, v_{i,4n}^2$ up to the last state $v_{i,4n}^2$ and then moves to the first state t_{n+i}^* in the fourth layer with remaining probability, as specified below. Furthermore, the construction ensures that the vertex degrees due to edges of probability p_2 and p_3 from the second to the third layer do not exceed 3.
- The single state t in the fourth layer of $M_{G,k}$ is replaced by a chain of $2n$ states t_1^*, \dots, t_{2n}^* , with a probability 1 self-loop at t_{2n}^* . For each $1 \leq i \leq n$, the last state $v_{i,4n+1}^3$ of the i -th chain in the third layer is connected to t_i^* by an edge of probability 1. For each $1 \leq i \leq n$, the last state $v_{i,4n}^2$ of the i -th chain in the second layer is connected to t_{n+i}^* by an edge of probability 1.

Note that $M_{G,k}^*$ contains $n^3 + 8n^2 + 5n$ states and transition probabilities are of size polynomial in n , therefore $M_{G,k}^*$ is polynomial in the size of G and k .

Correctness of reduction. While our modified construction ensures that the maximal vertex degree in the constructed Markov chain $M_{G,k}^*$ is equal to 3 and that $M_{G,k}^*$ remains acyclic, it preserves the key properties of $M_{G,k}$ about probabilities of moving between different chains. Therefore, one can follow the sequence of observations in the proof of Theorem 1 and analogously prove correctness of this modified reduction. Due to the proofs being analogous, we omit the details.

Computing Threshold Budgets in Discrete-Bidding Games

Guy Avni ✉

University of Haifa, Israel

Suman Sadhukhan ✉

University of Haifa, Israel

Abstract

In a two-player zero-sum graph game, the players move a token throughout the graph to produce an infinite play, which determines the winner of the game. *Bidding games* are graph games in which in each turn, an auction (bidding) determines which player moves the token: the players have budgets, and in each turn, both players simultaneously submit bids that do not exceed their available budgets, the higher bidder moves the token, and pays the bid to the lower bidder. We distinguish between *continuous-* and *discrete-*bidding games. In the latter, the granularity of the players' bids is restricted, e.g., bids must be given in cents. Continuous-bidding games are well understood, however, from a practical standpoint, discrete-bidding games are more appealing.

In this paper we focus on discrete-bidding games. We study the problem of finding *threshold budgets*; namely, a necessary and sufficient initial budget for winning the game. Previously, the properties of threshold budgets were only studied for reachability games. For parity discrete-bidding games, thresholds were known to exist, but their structure was not understood. We describe two algorithms for finding threshold budgets in parity discrete-bidding games. The first algorithm is a fixed-point algorithm, and it reveals the structure of the threshold budgets in these games. Second, we show that the problem of finding threshold budgets is in NP and coNP for parity discrete-bidding games. Previously, only exponential-time algorithms were known for reachability and parity objectives. A corollary of this proof is a construction of strategies that use polynomial-size memory.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Solution concepts in game theory

Keywords and phrases Discrete bidding games, Richman games, parity games, reachability games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.30

Related Version *Full Version*: <https://arxiv.org/abs/2210.02773>

Acknowledgements This research was supported in part by ISF grant no. 1679/21.

1 Introduction

Two-player zero-sum *graph games* are a central class of games. A graph game proceeds as follows. A token is placed on a vertex and the players move it throughout the graph to produce an infinite play, which determines the winner of the game. The central algorithmic problem in graph games is to identify the winner and to construct winning strategies. One key application of graph games is *reactive synthesis* [21], in which the goal is to synthesize a reactive system that satisfies a given specification no matter how the environment behaves.

Two orthogonal classifications of graph games are according to the *mode* of moving the token and according to the players' *objectives*. For the latter, we focus on two canonical qualitative objectives. In *reachability* games, there is a set of target vertices and Player 1 wins if a target vertex is reached. In *parity* games, each vertex is labeled with a parity index and an infinite path is winning for Player 1 iff the highest parity index that is visited infinitely often is odd. The simplest and most studied mode of moving is *turn-based*: the players alternate turns in moving the token.



© Guy Avni and Suman Sadhukhan;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 30; pp. 30:1–30:18



Leibniz International Proceedings in Informatics

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

We study *bidding graph games* [17, 16], which use the following mode of moving: both players have budgets, and in each turn, an auction (bidding) determines which player moves the token. Concretely, we focus on *Richman bidding* (named after David Richman): in each turn, both players simultaneously submit bids that do not exceed their available budget, the higher bidder moves the token, and pays his bid to the lower bidder. Note that the sum of budgets stays constant throughout the game. We distinguish between *continuous-* and *discrete-*bidding, where in the latter, the granularity of the players' bids is restricted. The central questions in bidding games revolve around the *threshold budgets*, which is a necessary and sufficient initial budget for winning the game.

Continuous-bidding games. Bidding games were largely studied under continuous bidding. We briefly survey the relevant literature. Bidding games were introduced in [17, 16]. The objective that was considered is a variant of reachability, which we call *double reachability*: each player has a target and a player wins if his target is reached (unlike reachability games in which Player 2's goal is to prevent Player 1 from reaching his target). It was shown that in continuous-bidding games, a target is necessarily reached, thus double-reachability games essentially coincide with reachability games under continuous-bidding.

Threshold budgets were shown to exist; namely, each vertex v has a value $\text{Th}(v)$ such that if Player 1's budget is strictly greater than $\text{Th}(v)$, he wins the game from v , and if his budget is strictly less than $\text{Th}(v)$, Player 2 wins the game. Moreover, it was shown that the threshold function Th is a *unique* function that satisfies the following property, which we call the *average property*. Suppose that the sum of budgets is 1, and t_i is Player i 's target, for $i \in \{1, 2\}$. Then, Th assigns a value in $[0, 1]$ to each vertex such that at the "end points", we have $\text{Th}(t_1) = 0$ and $\text{Th}(t_2) = 1$, and the threshold at every other vertex is the average of two of its neighbors. Uniqueness implies that the problem of finding threshold budgets¹ is in NP and coNP. Moreover, an intriguing equivalence was observed between reachability continuous-bidding games and a class of stochastic game [13] called *random-turn games* [20]. Intricate equivalences between *mean-payoff* continuous-bidding games and random-turn games have been shown in [6, 7, 8, 9] (see also [5]).

Parity continuous-bidding games were studied in [6]. The following key property was identified. Consider a strongly-connected parity continuous-bidding game \mathcal{G} . If the maximal parity index in \mathcal{G} is odd, then Player 1 wins with any positive initial budget, i.e., the thresholds in \mathcal{G} are all 0. Dually, if the maximal parity index in \mathcal{G} is even, then the thresholds are all 1. This property gives rise to a simple reduction from parity bidding games to double-reachability bidding games: roughly, a player's goal is to reach a bottom strongly-connected component in which he can win with any positive initial budget.

Discrete-bidding games. *Discrete-bidding games* are similar to continuous-bidding games only that we fix the sum of the budgets to be $k \in \mathbb{N}$ and bids are restricted to be integers. Ties in biddings need to be handled explicitly. We focus on the tie-breaking mechanism that was defined in [14]: one of the players has the *advantage* and when a tie occurs, the player with the advantage chooses between (1) use the advantage to win the bidding and pass it to the other player, or (2) keep the advantage and let the other player win. Other tie-breaking mechanisms and the properties that they lead to were considered in [1].

¹ Stated as a decision problem: given a game and a vertex v , decide whether $\text{Th}(v) \geq 0.5$.

The motivation to study discrete-bidding games is practical: in most applications, the assumption that bids can have arbitrary granularity is unrealistic. We point out that the results in continuous-bidding games, particularly those on infinite-duration games, do in fact develop strategies that bid arbitrarily small bids. It is highly questionable whether such strategies would be useful in practice.

Bidding games model ongoing and stateful auctions. Such auctions arise in various domains. An immediate example is auctions for online advertisements [19]. As another example, in *blockchain* technology, *miners* accept transaction fees, which can be thought of as bids, and prioritize transactions based on them. Verification against attacks is a well-studied problem [12, 4]. Bidding games have been applied as a mechanism for fair allocation of resources [18]. In addition, researchers have studied training of agents that accept “advice” from a “teacher”, where the advice is equipped with a “bid” that represents its importance [3]. Finally, recreation bidding games have been studied, e.g., bidding chess [11]. In all of these applications, the granularity of the bids is restricted.

Previous results. Threshold budgets and their properties were previously only studied for reachability discrete-bidding games [14]. In these games, discrete versions of the properties of continuous-bidding games were observed; namely, discrete versions of threshold budgets exist, they satisfy a discrete version of the average property, and, similar to continuous bidding, winning strategies are constructed in which a player’s bid is derived from the threshold budgets. A value-iteration algorithm was developed to compute the thresholds with a worst-case exponential running time, when k is given in binary.

Parity discrete-bidding games were shown to be determined in [1], but the structure of the threshold budgets was not understood. In particular, threshold budgets were not known to have the average property. The previously known algorithm to solve parity discrete-bidding games is naive. Construct an arena based on the exponentially-sized “configuration graph” that corresponds to a bidding game. Determinacy implies that it suffices to solve a turn-based game rather than a concurrent game on this arena. The essence of bidding games is completely lost in this construction; namely, the structure given by the threshold budgets is not used and the bids made by winning strategies have no connection with the thresholds. To make things worse, unlike reachability games, the properties of threshold budgets in parity discrete- and continuous-bidding games are known to differ significantly; namely, there are strongly-connected games in which the maximal parity index is odd and Player 1 loses with any initial budget, and no reduction is known to reachability discrete-bidding games.

Our results. We develop two complementary algorithms for computing threshold budgets in parity discrete-bidding games. Our first algorithm is a fixed-point algorithm. It is based on repeated calls to an algorithm to solve reachability discrete-bidding games in combination with a recursion over the parity indices, similar in spirit to Zielonka [22] and Kupferman and Vardi’s [15] algorithms to solve turn-based parity games. An important corollary from the algorithm is that threshold budgets in parity discrete-bidding games satisfy the average property. The worst-case running time of the algorithm is exponential.

Second, we show that the problem of finding threshold budgets in parity discrete-bidding games is in NP and coNP. The bound follows to reachability discrete-bidding games for which only an exponential-time algorithm was known. We briefly describe the idea of our proof. We first show that, interestingly, unlike continuous-bidding games, functions that satisfy the discrete average property are not unique, but by definition the threshold budgets are. Thus, one cannot simply guess a function and verify that it satisfies the average property. We

overcome this challenge as follows. Given a guess of function T , we first verify that it satisfies the average property. Then, we construct a partial bidding strategy f_T for Player 1 based on T , and construct a turn-based parity game in which a Player 1 strategy corresponds to a bidding strategy f' that agrees with f_T and a Player 2 strategy corresponds to a response to f' . We show that Player 1 wins the turn-based game iff T coincides with the threshold budgets. As a corollary, we show the existence of winning strategies that use memory that is polynomial in the size of the arena. Previously, only strategies that use exponential-sized memory were known.

2 Preliminaries

2.1 Concurrent games

We define the formal semantics of bidding games via two-player *concurrent games* [2]. Intuitively, a concurrent game proceeds as follows. A token is placed on a vertex of a graph. In each turn, both players concurrently select actions, and their joint actions determine the next position of the token. The outcome of a game is an infinite path. A game is accompanied by an objective, which specifies which plays are winning for Player 1. We focus on reachability and parity objectives, which we define later in this section.

Formally, a concurrent game is played on an *arena* $\langle A, V, \lambda, \delta \rangle$, where A is a finite non-empty set of actions, V is a finite non-empty set of vertices, the function $\lambda : V \times \{1, 2\} \rightarrow 2^A \setminus \{\emptyset\}$ specifies the allowed actions for Player i in vertex v , and the transition function is $\delta : V \times A \times A \rightarrow V$. Suppose that the token is placed on a vertex v and, for $i \in \{1, 2\}$, Player i chooses action $a_i \in \lambda(v)$. Then, the token moves to $\delta(v, a_1, a_2)$. For $u, v \in V$, we call u a *neighbor* of v if there is a pair of actions $\langle a_1, a_2 \rangle \in \lambda(v, 1) \times \lambda(v, 2)$ with $u = \delta(v, a_1, a_2)$. We denote the neighbors of v by $N(v) \subseteq V$.

A (finite) *history* is a sequence $\langle v_0, a_0^1, a_0^2 \rangle, \dots, \langle v_{n-1}, a_{n-1}^1, a_{n-1}^2 \rangle, v_n \in (V \times A \times A)^* \cdot V$ such that, for each $0 \leq i < n$, we have $v_{i+1} = \delta(v_i, a_i^1, a_i^2)$. A *strategy* is a function from histories to actions, thus it is of the form $\sigma : (V \times A \times A)^* \cdot V \rightarrow A$. We restrict attention to *legal* strategies; namely, strategies that for each history $\pi \in (V \times A \times A)^* \cdot V$ that ends in $v \in V$, choose an action in $\lambda(v, i)$, for $i \in \{1, 2\}$. A *memoryless* strategy is a strategy that, for every vertex v , assigns the same action to every history that ends in v .

Two strategies σ_1 and σ_2 for the two players and an initial vertex v_0 , give rise to a unique *play*, denoted $\text{play}(v_0, \sigma_1, \sigma_2)$, which is a sequence in $(V \times A \times A)^\omega$ and is defined inductively as follows. The first element of $\text{play}(v_0, \sigma_1, \sigma_2)$ is v_0 . Suppose that the prefix of length $j \geq 1$ of $\text{play}(v_0, \sigma_1, \sigma_2)$ is defined to be $\pi^j \cdot v_j$, where $\pi^j \in (V \times A \times A)^*$. Then, at turn j , for $i \in \{1, 2\}$, Player i takes action $a_i^j = \sigma_i(\pi^j \cdot v_j)$, the next vertex is $v^{j+1} = \delta(v_j, a_1^j, a_2^j)$, and we define $\pi^{j+1} = \pi^j \cdot \langle v_j, a_1^j, a_2^j \rangle \cdot v_{j+1}$. The *path* that corresponds to $\text{play}(v_0, \sigma_1, \sigma_2)$ is v_0, v_1, \dots

Turn-based games are a special case of concurrent games in which the vertices are partitioned between the two players. When the token is placed on a vertex v , the player who *controls* v decides how to move the token. Formally, a vertex v is controlled by Player 1 if for every action $a_1 \in A$, there is a vertex v' such that no matter Player 2 takes which action $a_2 \in A$, we have $v' = \delta(v, a_1, a_2)$. The definition is dual for Player 2. Note that a concurrent game that is not turn based might still contain some vertices that are controlled by one of the players.

2.2 Bidding games

A discrete-bidding game is played on an arena $\mathcal{G} = \langle V, E, k \rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, and $k \in \mathbb{N}$ is the sum of the players' budgets. For a vertex $v \in V$, we use $N(v)$ to denote its *neighbors*, namely $N(v) = \{u : E(v, u)\}$. The size of the arena is $O(|V| + |E| + \log(k))$.

Intuitively, in each turn, both players simultaneously choose a bid that does not exceed their available budgets. The higher bidder moves the token and pays the other player. Note that the sum of budgets stays constant throughout the game. Tie-breaking needs to be handled explicitly in discrete-bidding games as it can affect the properties of the game [1]. In this paper, we focus on the tie-breaking mechanism that was defined in [14]: exactly one of the players holds the *advantage* at every turn, and when a tie occurs, the player with the advantage chooses between (1) win the bidding and pass the advantage to the other player, or (2) let the other player win the bidding and keep the advantage.

Following [14], we denote the advantage with $*$. Let \mathbb{N} denote the non-negative integers and \mathbb{N}^* denote the set $\{0, 0^*, 1, 1^*, 2, 2^*, \dots\}$. Throughout the paper, we use k to denote the sum of budgets, and use $[k]$ to denote the set $\{0, 0^*, \dots, k, k^*\}$. Intuitively, when saying that Player 1 has a budget of $m^* \in [k]$, we mean that Player 1 can choose a bid in $\{0, \dots, m\}$ and that he has the advantage. Implicitly, we mean that Player 2's budget is $k - m$ and she does not have the advantage.

We define an order $<$ on \mathbb{N}^* by $0 < 0^* < 1 < 1^* < \dots$. Let $m \in \mathbb{N}^*$. We denote by $|m|$ the integer part of m , i.e., $|m^*| = m$. We define operators \oplus and \ominus over \mathbb{N}^* : $x^* \oplus y = (x + y)^*$, $x \oplus y = x + y$ for $x, y \in \mathbb{N}$. Intuitively, we use \oplus to keep track of the budget of a player when they loses a bidding to the other player, and as a result their budget gets increased by the other player's bid: therefore, in general $x^* \oplus y^*$ is not well-defined. The definition of \ominus is *almost* dual, we use the notation to keep track of the budget of a player when they wins a bidding: $x \ominus y = x - y$, $x^* \ominus y = (x - y)^*$, and in particular $x^* \ominus y^* = x - y$. For $B \in \mathbb{N}^*$, of exceptional use is $B \oplus 0^*$ and $B \ominus 0^*$, which respectively denote the *successor* and *predecessor* of B in \mathbb{N}^* according to $<$.

Consider an arena $\mathcal{G} = \langle V, E, k \rangle$ of a bidding game. We describe a concurrent game \mathcal{C} that corresponds to it. The vertices in \mathcal{C} consist of *configuration* vertices $C = V \times [k]$ and *intermediate* vertices $\{i_{c,b} : c \in C, b \leq k^*\}$. A vertex $c = \langle v, B \rangle \in (V \times [k])$ represents the configuration of the bidding game in which the token is placed on vertex $v \in V$, Player 1's budget is B , and Player 2's budget is $k^* \ominus B$. Consider a configuration vertex $c = \langle v, B \rangle$. The available actions for a player in c represent the legal bids and the vertex to move to upon winning. Thus, the available actions for Player 1 are $\{0, \dots, |B|\} \times N(v)$ and the available actions for Player 2 are $\{0, \dots, k - |B|\} \times N(v)$. Each intermediate vertex is owned by a single player, so we only specify their outgoing transitions below. We describe the transition function. Suppose that the token is placed on a configuration vertex $c = \langle v, B \rangle$ and Player i chooses action $\langle b_i, u_i \rangle$, for $i \in \{1, 2\}$. If $b_1 > b_2$, Player 1 wins the bidding and the game proceeds to $\langle u_1, B_1 \ominus b_1 \rangle$. The definition for $b_2 > b_1$ is dual. We address the case of a bidding tie, namely the case that $b_1 = b_2 = b$. Assume that Player 1 has the advantage, i.e., $c = \langle v, B_1^* \rangle$, and the other case is dual. The game proceeds to an intermediate vertex $i_{c,b}$ that is controlled by Player 1 and has two outgoing edges. The first edge models Player 1 using the advantage to win the bidding, and directs to the configuration $\langle u_1, B_1 - b_1 \rangle$. The second edge models Player 1 allowing Player 2 to win the bidding and keeping the advantage, and it directs to $\langle u_2, (B_1 + b_2)^* \rangle$. We often say that the player who holds the advantage bids b^* , and we mean that he bids b and uses the advantage if a tie occurs. Note that the size of the arena is $O(|V| \times k)$, which is exponential in the size of the bidding game.

Consider two strategies f and g in \mathcal{C} and an initial configuration $c = \langle v, B \rangle$. We often abuse notation and refer to $\tau = \text{play}(v, f, g)$ as the infinite path in \mathcal{G} that is obtained by removing intermediate vertices from π . We use $\text{inf}(\tau)$ to denote the set of vertices that τ visits infinitely often.

2.3 Objectives

An objective in a bidding game specifies the infinite paths that are winning for Player 1. We consider the following two canonical objectives:

- **Reachability** A game is equipped with a target set $T \subseteq V$. Player 1, the reachability player, wins an infinite play iff it visits T .
- **Parity** Each vertex is labeled by a *parity index*, given by a function $p : V \rightarrow \{1, \dots, d\}$, for $d \in \mathbb{N}$. A play τ is winning for Player 1 iff $\max_{v \in \text{inf}(\tau)} p(v)$ is odd.

In addition, we introduce the following extension of the two objectives above.

► **Definition 1** (Frugal objectives). *Consider an arena $\langle V, E, k \rangle$. A frugal objective is a set $S \subseteq V$ of sink states and a function $\mathbf{fr} : S \rightarrow [k]$ which assigns a frugal-target budget to each sink. Player 1's frugal objective is satisfied if the game reaches some $s \in S$ with Player 1's budget at least $\mathbf{fr}(s)$. We consider both frugal-reachability and frugal-parity games. Player 1 wins a frugal-reachability game if the frugal objective is satisfied. Note that a reachability game is a special case of a frugal-reachability game in which $\mathbf{fr} = 0$. A frugal-parity game is $\langle V, E, k, p, S, \mathbf{fr} \rangle$, where $p : (V \setminus S) \rightarrow \{0, \dots, d\}$. Player 1 wins a play π if (1) π does not reach S and satisfies the parity objective, or (2) π reaches S and satisfies the frugal objective.*

A winning strategy from a configuration $c = \langle v, B \rangle$ for Player 1 is a strategy f such that no matter which strategy g Player 2 chooses, $\text{play}(c, f, g)$ is winning for Player 1. We say that Player 1 *wins* from c if he has a winning strategy. The definition is dual for Player 2.

3 Threshold Budgets and the Average Property

A key quantity in bidding games is the *threshold budget* at a vertex, which intuitively represents the necessary and sufficient initial budget at that vertex for Player 1 to guarantee winning the game. It is formally defined as follows.

► **Definition 2** (Threshold budgets). *Consider a bidding game \mathcal{G} in which the sum of budgets is $k \in \mathbb{N}$. The threshold budget at a vertex v in \mathcal{G} , denoted $\text{Th}_{\mathcal{G}}(v)$, is such that if Player 1's budget at v is at least $\text{Th}_{\mathcal{G}}(v) \in [k]$, then Player 1 wins the game from v , and if his budget is at most $\text{Th}_{\mathcal{G}}(v) \ominus 0^*$, then Player 2 wins the game. We refer to the function $\text{Th}_{\mathcal{G}}$ as the threshold budgets.*

3.1 Reachability continuous-bidding games

The properties of threshold budgets in discrete-bidding games have only been studied for reachability games [14]. The properties of the threshold budgets and the techniques to prove them are similar to those used in reachability continuous-bidding games [17, 16]. We thus first survey the latter.

► **Definition 3** (Continuous threshold budgets). *Normalize the sum of budgets to 1. The continuous threshold budget at a vertex v is a budget $\text{Th}(v) \in [0, 1]$ such that if Player 1's budget exceeds $\text{Th}(v)$, he wins the game from v , and if Player 2's budget exceeds $1 - \text{Th}(v)$, she wins the game from v .*

We call the objective that was considered in [17, 16] *double-reachability*. Such a game is $\langle V, E, t_1, t_2 \rangle$, where for $i \in \{1, 2\}$, the vertex t_i is the target of Player i . The game ends once one of the targets is reached, and the player whose target is reached is the winner. Every other vertex has a path to both targets. Even though a priori, it is not implicit that one of the player always wins in a double reachability games (because the game might not reach either of the target vertices), here it is explicitly shown that this is not the case [17, 16].

► **Definition 4** (Continuous average property). *Consider a double-reachability continuous-bidding game $\mathcal{G} = \langle V, E, t_1, t_2 \rangle$ and a function $T : V \rightarrow [0, 1]$. We say that T has the continuous average property if $T(t_1) = 0$ and $T(t_2) = 1$, and for every other $v \in V \setminus \{t_1, t_2\}$, we have $T(v) = 0.5 \cdot (T(v^-) + T(v^+))$, where $v^+ := \arg \max_{u \in N(v)} \text{Th}(u)$ and $v^- := \arg \min_{u \in N(v)} \text{Th}(u)$.*

We show the main properties of double-reachability continuous-bidding games. The ideas used in the proof are adapted to the discrete setting in later sections.

► **Theorem 5** ([17, 16]). *Consider a double-reachability continuous-bidding game $\langle V, E, t_1, t_2 \rangle$. Continuous threshold budgets exist, and the threshold budgets $\text{Th} : V \rightarrow [0, 1]$ is the unique function that has the continuous average property.*

Proof Sketch. Let Th be a function that satisfies the continuous average property. We prove that $\text{Th}(v)$, for every vertex v , is the continuous threshold budget at v . Uniqueness follows immediately. We omit the proof of existence of Th .

Suppose that Player 1's budget at v is $\text{Th}(v) + \varepsilon$, for $\varepsilon > 0$. We describe a Player 1 winning strategy. Player 1 maintains the following invariant:

Invariant. When the token is on $u \in V$, Player 1's budget is strictly greater than $\text{Th}(u)$.

The invariant implies that Player 1 does not lose; indeed, it implies that if t_2 is reached, Player 1's budget is strictly greater than 1, which violates the assumption that the sum of budgets is 1.

The invariant holds initially, and we show how to maintain it. Suppose that the token is placed on $v \in V$ and Player 1's budget is $B = \text{Th}(v) + \varepsilon$. Recall that $v^+, v^- \in N(v)$ are respectively the neighbors of v with the maximal and minimal threshold budgets. Let $b = 0.5 \cdot (\text{Th}(v^+) - \text{Th}(v^-))$. The key observation is that $B + b = \text{Th}(v^+) + \varepsilon$ and $B - b = \text{Th}(v^-) + \varepsilon$. We claim that by bidding b , Player 1 guarantees that the invariant is maintained. Indeed, if he wins the bidding, he moves the token to v^+ , and if he loses the bidding, the worst that Player 2 can do is move the token to v^- .

We omit the details of how Player 1 guarantees winning, i.e., forcing the token to t_1 .

Finally, we show that Player 2 wins when Player 1's budget is $\text{Th}(v) - \varepsilon$. We intuitively “flip” the game and associate Player 1 with Player 2. Let \mathcal{G}' be the same as \mathcal{G} only that Player 1's goal is to reach t_2 and Player 2's goal is to reach t_1 . For every $u \in V$, define Th' as $\text{Th}'(u) = 1 - \text{Th}(u)$. A key observation is that Th' satisfies the average property in \mathcal{G}' . Now, in order to win from v in \mathcal{G} when Player 1's budget is $\text{Th}(v) - \varepsilon$, Player 2 follows a winning Player 1 strategy in \mathcal{G}' with an initial budget of $1 - \text{Th}(v) + \varepsilon$. ◀

3.2 Constructing strategies based on the discrete average property

We first adapt the definition of the average property (Def. 4) to the discrete setting.

► **Definition 6** (Average property). Consider a discrete-bidding game $\mathcal{G} = \langle V, E, k, S, \mathbf{fr} \rangle$ with frugal objective. We say that a function $T : V \rightarrow [k] \cup \{k+1\}$ has the average property if for every $s \in S$, we have $T(s) = \mathbf{fr}(s)$, and for every $v \in V \setminus S$,

$$T(v) = \lfloor \frac{|T(v^+)| + |T(v^-)|}{2} \rfloor + \varepsilon,$$

$$\text{where } \varepsilon = \begin{cases} 0 & \text{if } |T(v^+)| + |T(v^-)| \text{ is even and } T(v^-) \in \mathbb{N} \\ 1 & \text{if } |T(v^+)| + |T(v^-)| \text{ is odd and } T(v^-) \in \mathbb{N}^* \setminus \mathbb{N} \\ * & \text{otherwise} \end{cases}$$

where $v^+ := \arg \max_{u \in N(v)} Th(u)$ and $v^- := \arg \min_{u \in N(v)} Th(u)$

Note that, the range of T includes $k+1$, which captures the threshold budget of a player at a vertex from where the other player wins the game with budget 0. Consider a function $T : V \rightarrow [k] \cup \{k+1\}$ that satisfies the average property. We develop a *partial strategy* f_T , which is a function from histories to $[k] \times 2^V$. An output $\langle b, A \rangle$ of f_T means that the bid is b and upon winning, Player 1 must choose an *allowed vertex* in A to move the token to. A strategy f' that *agrees* with f_T bids in the same manner and upon winning a bidding, it chooses a vertex in A .

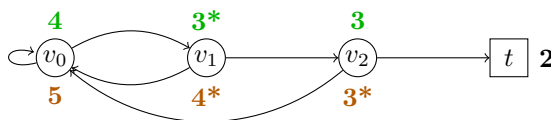
We define f_T so that when Player 1 plays according to a strategy that agrees with f_T , an invariant is maintained on Player 1's budget, similar to the invariant in the continuous setting (see the proof of Thm. 5). Suppose that a history ends in a vertex v with Player 1's budget $B \geq T(v)$. Intuitively, as in Thm. 5, when Player 1 wins the bidding in v , he proceeds to some neighbor v^- that attains the minimal value according to T , and when Player 2 wins the bidding, the worst she can do is move to a vertex v^+ that attains the maximal value according to T . Formally, f_T restricts the choice of neighbor of v to a set of *allowed vertices*, denoted $A(v)$, and depend only v . Let $v^+ = \arg \max_{u \in N(v)} T(u)$ and $v^- = \arg \min_{u \in N(v)} T(u)$. We define $A(v) = \{u \in N(v) : T(u) = T(v^-)\}$ when $T(v^-) \in \mathbb{N}$, and $A(v) = \{u \in N(v) : T(u) \leq T(v^-) \oplus 0^*\}$ when $T(v^-) \in \mathbb{N}^* \setminus \mathbb{N}$.

Next, for each $v \in V$, we define a bid that f_T proposes. Suppose that Player 1's budget is $B \in [k]$, for $B \geq T(v)$. We define f_T so that it proposes one of two possible bids b_v^T or $b_v^T \oplus 0^*$, depending on whether Player 1 holds the advantage. Intuitively, Player 1 "attempts" to bid b_v^T at v . This is not possible if $b_v^T \in \mathbb{N}^* \setminus \mathbb{N}$ and $B \in \mathbb{N}$, i.e., Player 1 wants to use the advantage but does not have it. In such a case, Player 1 bids $b_v^T \oplus 0^* \in \mathbb{N}$. Formally, we define

$$b_v^T = \begin{cases} \frac{|T(v^+)| - |T(v^-)|}{2} & \text{When } |T(v^+)| + |T(v^-)| \text{ is even and } T(v^-) \in \mathbb{N} \\ \lfloor \frac{|T(v^+)| - |T(v^-)|}{2} \rfloor & \text{When } |T(v^+)| + |T(v^-)| \text{ is odd and } T(v^-) \in \mathbb{N}^* \setminus \mathbb{N} \\ \frac{|T(v^+)| - |T(v^-)|}{2} \oplus 0^* & \text{When } |T(v^+)| + |T(v^-)| \text{ is even and } T(v^-) \in \mathbb{N}^* \setminus \mathbb{N} \\ \lfloor \frac{|T(v^+)| - |T(v^-)|}{2} \rfloor \oplus 0^* & \text{When } |T(v^+)| + |T(v^-)| \text{ is odd and } T(v^-) \in \mathbb{N} \end{cases} \quad (1)$$

Assuming that the game reaches v with a budget of $B \geq T(v)$, we define f_T to bid $b^T(v, B)$ where $b^T(v, B) = b_v^T$ when both b_v^T and B belong to either \mathbb{N} or $\mathbb{N}^* \setminus \mathbb{N}$, and $b_v^T \oplus 0^*$ otherwise. We formalize the guarantees of f_T in the lemma below, whose proof can be found in the full version[10].

► **Lemma 7.** For every $v \in V$ and $B \geq T(v)$, we have $B \oplus b^T(v, B) \geq T(v^-)$ and $B \oplus b^T(v, B) \oplus 0^* \geq T(v^+)$.



■ **Figure 1** A discrete-bidding reachability game with two functions that satisfy the average property.

A corollary of Lem. 7 is that any strategy that agrees with f_T maintains an invariant on Player 1’s budget and is thus a legal strategy, i.e., it never prescribes bids that exceed the available budget.

► **Corollary 8.** *Suppose that Player 1 plays according to a strategy that agrees with f_T starting from configuration $\langle u, B \rangle$ having $B \geq T(u)$, we have:*

- *When the game reaches $v \in V$, Player 1’s budget is at least $T(v)$.*
- *The bid b prescribed by f_T does not exceed the available budget, i.e., $b \leq B$.*

3.3 Properties of functions with the average property

We show that, somewhat surprisingly, unlike in continuous-bidding, functions that satisfy the discrete average property are not unique. That is, there are functions that satisfy the average property but do not coincide with the threshold budgets.

► **Theorem 9.** *The reachability discrete-bidding game \mathcal{G}_1 that is depicted in Fig. 1 with target t for Player 1 has more than one function that satisfies the average property.*

Proof. Assume a total budget of $k = 5$. We represent a function $T : V \rightarrow [k]$ as a vector $\langle T(v_0), T(v_1), T(v_2), T(t) \rangle$. It is not hard to verify that both $\langle 4, 3^*, 3, 2 \rangle$ and $\langle 5, 4^*, 3^*, 2 \rangle$ satisfy the average property. (The latter represents the threshold budgets). ◀

The following lemma, whose proof can be found in the full version [10], is key in developing a winning Player 2 strategy. We intuitively show that the “complement” of T satisfies the average property. The idea is similar to the continuous case (see the last point in the proof of Thm. 5).

► **Lemma 10.** *Let $\mathcal{G} = \langle V, E, k, S, fr \rangle$ be a discrete-bidding game with a frugal objective. Let $T : V \rightarrow [k] \cup \{k + 1\}$ be a function that satisfies the average property. We define $T' : V \rightarrow [k] \cup \{k + 1\}$ as follows. For $v \in V$, let $T'(v) = k^* \ominus (T(v) \ominus 0^*)$ when $T(v) > 0$, and $T'(v) = k + 1$ otherwise.*

Then, T' satisfies the average property.

3.4 Frugal-reachability discrete-bidding games

We close this section by extending the results of [14] from reachability to frugal-reachability discrete-bidding games.

► **Lemma 11.** *Consider a frugal-reachability discrete-bidding game $\mathcal{G} = \langle V, E, k, S, fr \rangle$. If $T : V \rightarrow [k] \cup \{k + 1\}$ is a function that satisfies the average property, then $T(v) \leq \text{Th}_{\mathcal{G}}(v)$ for every $v \in V$.*

Proof. We show that if for some vertex v , Player 1 has a budget less than $T(v)$, then Player 2 has a winning strategy, which proves that the threshold budgets for Player 1 cannot be less than $T(v)$, when T is a average property satisfying function.

30:10 Computing Threshold Budgets in Discrete-Bidding Games

Given such T that satisfies the average property, we construct T' as in Lem. 10. Let $\langle v, B_1 \rangle$ be a configuration, where $v \in V$, Player 1's budget is B_1 , and implicitly, Player 2's budget is $B_2 = k^* \ominus B_1$. Note that $B_1 < T(v)$ iff $B_2 \geq T'(v)$. Moreover, for every $s \in S$, we have $T'(s) = k^* \ominus (\mathbf{fr}(s) \ominus 0^*)$. We “flip” the game; namely, we associate Player 2 with Player 1, and construct a partial strategy $f_{T'}$ for Player 2 as in Sec. 3.2. We construct a Player 2 strategy f' that agrees with $f_{T'}$: for each $v \in V$, we arbitrarily choose a neighbor u from the allowed vertices. By Corollary 8, no matter how Player 1 responds, whenever the game reaches $\langle u, B_1 \rangle$, we have $B_2 \geq T'(u)$. The invariant implies that f' is a winning strategy. Indeed, if the game does not reach a sink, Player 2 wins, and if it does, Player 1's frugal objective is not satisfied. ◀

► **Lemma 12.** *Consider a frugal-reachability discrete-bidding game $\mathcal{G} = \langle V, E, k, S, \mathbf{fr} \rangle$. There is a function T that satisfies the average property with $T(v) \geq \text{Th}_{\mathcal{G}}(v)$, for every $v \in V$.*

Proof. The proof is similar to the one in [14]. We illustrate the main ideas. For $n \in \mathbb{N}$, we consider the *truncated game* $\mathcal{G}[n]$, which is the same as \mathcal{G} only that Player 1 wins iff he wins in at most n steps. We find a sufficient budget for Player 1 to win in the vertices in $\mathcal{G}[n]$ in a backwards-inductive manner. For the base case, for every vertex $u \in V$, since Player 1 cannot win from u in 0 steps, we have $T_0(u) = k + 1$. For $s \in S$, we have $T_0(s) = \mathbf{fr}(s)$. Clearly, $T_0 = \text{Th}_{\mathcal{G}[0]}$. For the inductive step, suppose that T_{n-1} is computed. For each vertex v , we define $T_n(v) = \lfloor \frac{|T_{n-1}(v^+)| + |T_{n-1}(v^-, k)|}{2} \rfloor + \varepsilon$ as in Def. 6. Following a similar argument to Thm. 5, it can be shown that if Player 1's budget is $T_n(v)$, he can bid b so that if he wins the bidding, his budget is at least $T_{n-1}(v^-)$ and if he loses the bidding, his budget is at least $T_{n-1}(v^+)$. By induction we get $\text{Th}_{\mathcal{G}[n]}(v) = T_n(v)$, for every $v \in V$. For every vertex v , let $T(v) = \lim_{n \rightarrow \infty} T_n(v)$. It is not hard to show that T satisfies the average property and that $T(v) \geq \text{Th}_{\mathcal{G}}(v)$, for every $v \in V$. ◀

Let T be a function that results from the fixed-point computation from the proof of Lem. 12. Since it satisfied the average property, we apply Lem. 11 to show that Player 2 wins from v when Player 1's budget is $T(v) \ominus 0^*$. We thus conclude the following.

► **Theorem 13.** *Consider a frugal-reachability discrete-bidding game $\mathcal{G} = \langle V, E, k, S, \mathbf{fr} \rangle$. Threshold budgets exist and satisfy the average property. Namely, there exists a function $T : V \rightarrow [k] \cup \{k + 1\}$ such that for every vertex $v \in V$*

- *if Player 1's budget is $B \geq T(v)$, then Player 1 wins the game, and*
- *if Player 1's budget is $B < T(v)$, then Player 2 wins the game*

Moreover, there is an exponential-time algorithm for finding such a T .

4 A Fixed-Point Algorithm for Finding Threshold Budgets

In this section, we develop a fixed-point algorithm for finding threshold budgets in frugal-parity discrete-bidding games. As a corollary, we show, for the first time, that threshold budgets in parity discrete-bidding games satisfy the average property.

For the remainder of this section, fix a frugal-parity game $\mathcal{G} = \langle V, E, k, p, S, \mathbf{fr} \rangle$. Denote the maximal parity index by $d \in \mathbb{N}$ and let $F_d = \{v : p(v) = d\}$. For a bidding game \mathcal{G} , instead of $\text{Th}_{\mathcal{G}}$, we sometimes use $\text{FrRe-Th}_{\mathcal{G}}$ and $\text{FrPa-Th}_{\mathcal{G}}$ to highlight that \mathcal{G} is respectively a frugal-reachability and frugal-parity game.

A description of the algorithm

The algorithm recurses over the parity indices. The base case is when only one parity index is used. Then, \mathcal{G} is a frugal-reachability game and we use the algorithm in Thm. 13.

For the induction step, suppose that $d > 1$. We describe the key idea. For ease of presentation, we assume that the maximal parity index d is even and we describe the algorithm from Player 1's perspective. The definition for an odd d is dual from Player 2's perspective. Since d is even, in order for Player 1 to win, it is necessary (but not sufficient) to visit F_d only finitely often.

We iteratively define and solve a sequence of frugal-parity games $\mathcal{G}_0, \mathcal{G}_1, \dots$. For $i \geq 0$, the arena of \mathcal{G}_i is obtained from \mathcal{G} by setting F_d to be sinks. The games differ in the frugal target in the “new” sinks F_d . We set the frugal target budgets in \mathcal{G}_i so that, for every vertex v that is not a sink, $\text{Th}_{\mathcal{G}_i}(v)$ is a necessary and sufficient initial budget for winning in \mathcal{G} by visiting F_d at most i times. Since \mathcal{G}_i has only $d - 1$ parity indices, we solve it recursively.

The definition of the frugal target budgets in \mathcal{G}_0 is immediate: since no visits to F_d are allowed, simply set the frugal target budget to be $k + 1$ in these vertices. Since the sum of budgets is k , a play that ends in F_d in \mathcal{G}_0 is necessarily losing for Player 1, thus in order to win, he must satisfy the parity or frugal objective without visiting F_d .

In order to define the frugal target budgets in \mathcal{G}_1 , we first construct a frugal-reachability game from \mathcal{G} , which we denote \mathcal{R}_0 . The sinks in \mathcal{R}_0 are $V \setminus F_d$. The frugal target budget at $u \in (V \setminus F_d)$ is defined to be $\text{fr}(u) = \text{Th}_{\mathcal{G}_0}(u)$. Thus, when the game starts at $v \in F_d$ with a Player 1 budget of $\text{Th}_{\mathcal{R}_0}(v)$, Player 1 can guarantee that the game eventually reaches $V \setminus F_d$ with a budget that suffices for winning in \mathcal{G} without visiting F_d again.

We define the frugal target budgets in \mathcal{G}_1 . For each $v \in F_d$ we define $\text{fr}(v) = \text{Th}_{\mathcal{R}_0}(v)$. Then, when \mathcal{G} starts from $u \in (V \setminus F_d)$ with a Player 1 budget of $\text{Th}_{\mathcal{G}_1}(u)$, Player 1 plays as follows. He first follows a winning strategy in \mathcal{G}_1 to ensure either (1) the parity condition is satisfied, (2) an “old” sink $s \in S$ is reached with budget at least $\text{fr}(s)$, or (3) the game reaches $v \in F_d$ with a budget of at least $\text{Th}_{\mathcal{R}_0}(v)$. Cases (1)-(2) are winning in \mathcal{G} . In Case (3), Player 1 plays as described above to guarantee winning in \mathcal{G} without visiting F_d again. The construction of $\mathcal{R}_1, \mathcal{R}_2, \dots$ and $\mathcal{G}_2, \mathcal{G}_3, \dots$ follows the same idea.

Formally, for $i \geq 0$, we define $\mathcal{G}_i = \langle V \setminus F_d, E', p', S \cup F_d, \text{fr}_i \rangle$, where E' is obtained from E by removing outgoing edges from vertices in F_d , i.e., $E'(v, u)$ iff $E(v, u)$ and $v \notin F_d$, the parity function p' coincides with p but is not defined over F_d , and fr_i is defined below. Note that p' assigns at most $d - 1$ parity indices. The function fr_i coincides with fr on the “old” sinks; namely, $\text{fr}_i(s) = \text{fr}(s)$, for every $s \in S$.

For the “new” sinks F_d , for $i \geq 0$, the definition fr_i is inductive. Let $v \in F_d$. We define $\text{fr}_0(v) = k + 1$, meaning that Player 1 is not allowed to visit F_d at all in \mathcal{G}_0 . For $i \geq 1$, assume that fr_i has been defined and we define fr_{i+1} as follows. Since \mathcal{G}_i has less parity indices than \mathcal{G} , we can recursively run the algorithm to obtain $\text{FrPa-Th}_{\mathcal{G}_i}(u)$, for each $u \in V \setminus F_d$. As we prove formally below, a budget of $\text{FrPa-Th}_{\mathcal{G}_i}(v)$ suffices for Player 1 to win \mathcal{G} while visiting F_d only i times. For $i \geq 0$, we construct a frugal-reachability game $\mathcal{R}_i = \langle V, E'', V \setminus F_d, \text{FrPa-Th}_{\mathcal{G}_i} \rangle$, where E'' is obtained from E by removing outgoing edges from every vertex in $(V \setminus F_d)$. That is, in \mathcal{R}_i , the only vertices that are *not* sinks are the vertices in F_d , and in order to win in a sink $u \in (V \setminus F_d)$, Player 1's budget should be at least $\text{FrPa-Th}_{\mathcal{G}_i}(u)$. We can now define the the frugal target fr_{i+1} of \mathcal{G}_{i+1} : for each $v \in F_d$, define $\text{fr}_{i+1}(v) = \text{FrRe-Th}_{\mathcal{R}_i}(v)$.

The algorithm is described Alg. 1 for an even d and from Player 1's perspective.

Algorithm 1 Frugal-Parity-Threshold(\mathcal{G}).

```

if  $\mathcal{G}$  uses one parity index then
  Return Frugal-Reachability-Threshold( $\mathcal{G}$ )
 $\mathbf{fr}_0(v) = k + 1$ , for  $v \in F_d$ 
for  $i = 0, 1, \dots$  do
   $\mathbf{FrPa-Th}_{\mathcal{G}_i} \leftarrow$  Frugal-Parity-Threshold( $\mathcal{G}_i$ )
   $\mathbf{FrRe-Th}_{\mathcal{R}_i} \leftarrow$  Frugal-Reachability-Threshold( $\mathcal{R}_i$ )
  For each  $v \in F_d$ , define  $\mathbf{fr}_{i+1}(v) = \mathbf{FrRe-Th}_{\mathcal{R}_i}(v)$ 
  if  $\mathbf{fr}_i(v) = \mathbf{fr}_{i+1}(v)$ , for all  $v \in F_d$  then
    Define  $\mathbf{FrPa-Th}_{\mathcal{G}}(v) = \mathbf{fr}_i(v)$  for  $v \in F_d$ 
    Define  $\mathbf{FrPa-Th}_{\mathcal{G}}(u) = \mathbf{FrRe-Th}_{\mathcal{G}_i}(u)$  for  $u \in V \setminus F_d$ .
  Return  $\mathbf{FrPa-Th}_{\mathcal{G}}$ 

```

Correctness

The intuition for the proof of the following lemma is described above, and we formally prove this below. The formal proof can be found in the full version [10].

► **Lemma 14.** *Let d be the maximal parity index in \mathcal{G} . Define $j = 1$ when d is even and $j = 2$ when d is odd. For $i \geq 0$, let $\mathbf{FrPa-Th}_{\mathcal{G}_i}$ be the threshold budget of Player j in the game \mathcal{G}_i . Then, for $v \in V \setminus F_d$, a budget of $\mathbf{FrPa-Th}_{\mathcal{G}_i}(v)$ suffices for Player j to win \mathcal{G} while visiting F_d at most i times.*

It follows from the following lemma, whose proof can be found in the full version [10], that Alg. 1 reaches a fixed point and terminates.

► **Lemma 15.** *For every $v \in F_d$ and $i \geq 0$, we have $\mathbf{fr}_i(v) \geq \mathbf{fr}_{i+1}(v)$.*

Next, we show that the budgets returned by Alg. 1 are necessary for Player 1 to win.

► **Lemma 16.** *Consider an output T of Algorithm 1 and let $u \in V \setminus F_d$. If Player 1's budget is $T(u) \ominus 0^*$, then Player 2 wins \mathcal{G} from v .*

Proof. The proof is by induction on the number of parity indices in \mathcal{G} . When there is one parity index, \mathcal{G} is a frugal-reachability game and the proof follows from Thm. 13. For the induction step, let $i \geq 0$ be the index at which the algorithm reaches a fixed point. That is, for every $u \in V \setminus F_d$, we have $T(u) = \mathbf{FrPa-Th}_{\mathcal{G}_i}(u)$. Since \mathcal{G}_i is a game with less parity indices than \mathcal{G} , by the induction hypothesis, a budget of $\mathbf{FrPa-Th}_{\mathcal{G}_i}(u)$ is necessary for Player 1 to win from u in \mathcal{G}_i . That is, if Player 1's budget is less than $\mathbf{FrPa-Th}_{\mathcal{G}_i}(u)$, then Player 2 has a strategy that guarantees that an infinite play satisfies Player 2's parity objective, and a finite play that ends in $v \in S \cup F_d$ violates Player 1's frugal objective $\mathbf{fr}_i(v)$.

We construct a winning Player 2 strategy in \mathcal{G} . Player 2 initially follows a winning strategy in \mathcal{G}_i . Assume that Player 1 plays according to some strategy and let π be the resulting play. If π is infinite or ends in S , then π is a play in \mathcal{G} and is thus winning for Player 2 in both games. Suppose that π ends in $v \in F_d$. Player 2's strategy guarantees that Player 1's budget at v is at most $\mathbf{fr}_i(v) \ominus 0^* = \mathbf{FrRe-Th}_{\mathcal{R}_i}(v) \ominus 0^*$. Suppose that Player 2 follows a winning strategy from v in \mathcal{R}_i , Player 1 follows some strategy, and let π' be the resulting play. Since Player 2's strategy is winning, there are two cases. First, π' remains in F_d , thus π' is a play in \mathcal{G} that is winning for Player 2 since d is even. Second, π' ends in a vertex $u' \in V \setminus F_d$ is reached with Player 1's budget at most $\mathbf{FrPa-Th}_{\mathcal{G}_{i-1}}(u') \ominus 0^*$. Note that since the algorithm terminates at a fixed point, we have $\mathbf{FrPa-Th}_{\mathcal{G}_{i-1}}(u') = \mathbf{FrPa-Th}_{\mathcal{G}_i}(u')$.

Player 2 switches to a winning strategy in \mathcal{G}_i and repeats. Note that an infinite play in which Player 2 alternates infinitely often between a winning strategy in \mathcal{G}_i and a winning strategy in \mathcal{R}_i necessarily visits F_d infinitely often and is thus winning for Player 2 in \mathcal{G} . ◀

We conclude with the following theorem. Proving that threshold budgets satisfy the average property is done by induction on the parity indices.

► **Theorem 17.** *Given a frugal-parity discrete-bidding game \mathcal{G} , Alg. 1 terminates and returns $\text{Th}_{\mathcal{G}}$. Moreover, $\text{Th}_{\mathcal{G}}$ satisfies the average property.*

5 Finding threshold budgets is in NP and coNP

We consider the following decision problem.

► **Definition 18** (Finding threshold budgets). *Given a bidding game $\mathcal{G} = \langle V, E, k \rangle$, a vertex $v \in V$, and $\ell \in [k]$, decide whether $\text{Th}_{\mathcal{G}}(v) \geq \ell$.*

Consider a frugal-parity discrete-bidding game \mathcal{G} with vertices V and consider a function $T : V \rightarrow [k] \cup \{k+1\}$. We describe a polynomial-time algorithm to check whether $T = \text{Th}_{\mathcal{G}}$. Given such an algorithm, it is not hard to show that the problem of finding threshold budgets is in NP and coNP. Indeed, given \mathcal{G} , a vertex v , and $\ell \in [k]$, guess T and verify, using the algorithm, that $T = \text{Th}_{\mathcal{G}}$. Then, check whether $T(v) \geq \ell$, and answer accordingly.

Our algorithm is based on a reduction to turn-based games. We first verify that T satisfies the average property, and if it does not, we reject, following Thm. 17. Next, we construct the partial strategy f_T based on T as in Sec. 3.2. Recall that given a history that ends in v , the function f_T prescribes a bid and a set $A(v) \subseteq V$ of allowed vertices for Player 1 to choose from upon winning the bidding. A strategy f' agrees with f_T if it bids in the same manner and always chooses only allowed vertices. In order to verify whether T is a correct guess, i.e., $T = \text{Th}_{\mathcal{G}}$, we construct a parity turn-based game $G_{T,\mathcal{G}}$ in which a Player 1 strategy corresponds to a strategy f' that agrees with f in \mathcal{G} and a Player 2 strategy corresponds to a response to f' in \mathcal{G} . We show that the procedure is sound and complete; namely, if Player 1 wins in $G_{T,\mathcal{G}}$, he wins in \mathcal{G} (hence $T \geq \text{Th}_{\mathcal{G}}$), and if $T \geq \text{Th}_{\mathcal{G}}$, then Player 1 wins in $G_{T,\mathcal{G}}$. Finally, in order to verify that $T \leq \text{Th}_{\mathcal{G}}$, we apply the same procedure from Player 2's perspective.

5.1 From bidding games to turn-based games

Consider a function T that satisfies the average property, and let f_T be the partial strategy as constructed in Sec. 3.2. We construct a parity turn-based game $G_{T,\mathcal{G}}$ such that if Player 1 wins in every vertex in $G_{T,\mathcal{G}}$, then $T \geq \text{Th}_{\mathcal{G}}$.

Intuitively, $G_{T,\mathcal{G}}$ simulates \mathcal{G} . In each turn, Player 1's bid is determined according to f_T . Suppose that Player 1 bids b . Player 2's actions in $G_{T,\mathcal{G}}$ represent responses to b . Namely, Player 2 can choose between winning the bidding by bidding $b \oplus 0^*$ and in addition choosing the successor vertex in \mathcal{G} , or losing the bidding by bidding 0 and letting Player 1 decide how the game proceeds.

A key challenge is keeping the size of $G_{T,\mathcal{G}}$ polynomial in the size of \mathcal{G} . Consider a history that ends in a configuration $\langle v, B \rangle$, for $B \geq T(v)$. Recall that f_T can prescribe one of two possible bids: the bid that f_T prescribes at $\langle v, B \rangle$ coincides either with the bid that it prescribes in $\langle v, T(v) \rangle$ or $\langle v, T(v) \oplus 0^* \rangle$, depending on which of the two agrees with B on which player has the advantage. We obtain a polynomial-sized arena by representing every configuration $\langle v, B \rangle$, for $B > T(v) \oplus 0^*$, with a vertex $\langle v, \top \rangle$.

See an example of the construction in the App. A.

Formally, we define $G_{T,\mathcal{G}} = \langle V_1, V_2, E, \gamma \rangle$, where for $i \in \{1, 2\}$, the vertices V_i are controlled by Player i , $E \subseteq (V_1 \cup V_2) \times (V_1 \cup V_2)$ is a collection of edges, and $\gamma : V_1 \cup V_2 \rightarrow \{1, 2, \dots, d\}$ assigns parity indices to the vertices. The vertices of $G_{T,\mathcal{G}}$ are $V_2 = \{\langle v, T(v) \rangle, \langle v, T(v) \oplus 0^* \rangle, \langle v, \top \rangle : v \in (V \cup S)\}$ and $V_1 = \{w^1 : w = \langle v, B \rangle \in V_2, B \neq \top\}$. We define the edges in $G_{T,\mathcal{G}}$. The sinks in $G_{T,\mathcal{G}}$ are of the form $\langle s, B \rangle$, for $s \in S$, or $\langle v, \top \rangle$, for $v \in V$. Sinks have self loops. We describe the other edges. Let $w = \langle v, B \rangle \in V_2$, where $B \neq \top$ and $v \notin S$. Intuitively, reaching w in $G_{T,\mathcal{G}}$ means that \mathcal{G} is in configuration $\langle v, B_1 \rangle$, where $B_1 \geq B$ and they agree on which player has the advantage. Thus, f_T bids $b^T(v, B)$. Outgoing edges from w model Player 2's two options. First, Player 2 can choose to win the bidding by bidding $b^T(v, B) \oplus 0^*$ (any higher bid is wasteful on her part) and moving the token to $u \in N(v)$. The next configuration is intuitively $\langle u, B' \rangle$, where $B' = B \oplus b^T(v, B) \oplus 0^*$. If $T(u) \leq B' \leq T(u) \oplus 0^*$, then $\langle u, B' \rangle \in V_2$ and we define $E(\langle v, B \rangle, \langle u, B' \rangle)$. Otherwise, we truncate Player 1 budget by defining $E(\langle v, B \rangle, \langle u, \top \rangle)$. We disallow transitions in which Player 2's bid exceeds her available budget, i.e., when $b^T(v, B) \oplus 0^* > k^* \ominus B$. Second, Player 2 can choose to move to w^1 modeling Player 2 allowing Player 1 to win the bidding, e.g., by bidding 0. In this case, Player 1's budget is updated to $B' = B \ominus b^T(v, B)$ and he moves the token to some vertex in $A(u)$. Thus, the neighbors of w^1 are $\langle u, B' \rangle$, for $u \in A(v)$. By the proof of Corollary 8, $B' \in \{T(u), T(u) \oplus 0^*\}$, thus $\langle u, B' \rangle \in V_2$. Note that all paths in $G_{T,\mathcal{G}}$ are infinite. Finally, we define the parity indices. A non-sink vertex in $G_{T,\mathcal{G}}$ "inherits" its parity index from the vertex in \mathcal{G} ; namely, for $w = \langle v, B \rangle \in V_2$, we define $\gamma(w) = \gamma(w^1) = p(v)$. We define γ so that Player 1 wins in sinks, thus we set the parity index of a sink to be odd.

5.2 Correctness

In this section, we prove soundness and completeness of the approach. We start with soundness.

► **Lemma 19.** *If Player 1 wins from every vertex in $G_{T,\mathcal{G}}$, then $T \geq \text{Th}_{\mathcal{G}}$.*

Proof. We describe the main ideas of the proof and the details can be found in the full version [10]. Assume that Player 1 wins from every vertex in $G_{T,\mathcal{G}}$ and fix some memoryless winning strategy f' . We describe a winning strategy f^* of Player 1 in \mathcal{G} , which he can play when he has a budget of at least $T(v)$ at vertex v for all vertices v of \mathcal{G} . This proves that $T \geq \text{Th}_{\mathcal{G}}$.

Suppose that \mathcal{G} starts from $c_0 = \langle v, T(v) \rangle$. We initiate $G_{T,\mathcal{G}}$ from $v_0 = \langle v, T(v) \rangle$. Suppose that Player 2 plays according to a strategy g^* in \mathcal{G} . Player 1 simulates a Player 2 strategy g in $G_{T,\mathcal{G}}$ so that when \mathcal{G} reaches a configuration $c = \langle u, B_1 \rangle$, the vertex in $G_{T,\mathcal{G}}$ is $w = \langle u, B \rangle$, where $B \in \{T(u), T(u) \oplus 0^*\}$ agrees with B_1 on the advantage. We describe how we simulate g^* with g , and how f^* simulates f' . Suppose that \mathcal{G} is in configuration $c = \langle u, B_1 \rangle$ and $G_{T,\mathcal{G}}$ is in $w = \langle u, B \rangle$. We define f^* to agree with f_T and bid $b_T(u, B)$. If g^* loses the bidding, then we define g to proceed to w^1 and we define f^* to match the move of f' from w^1 . If g^* wins the bidding in \mathcal{G} , we define g to win the bidding and move the token as g^* does. In the full version [10], we show that the correspondence between the games is maintained.

Let π be the play in $G_{T,\mathcal{G}}$ that results from f' and g and π^* the play in \mathcal{G} that results from f^* and g^* . Since f' is winning, π satisfies Player 1's objective. We distinguish between three cases. In the first case, π does not reach a sink in $G_{T,\mathcal{G}}$. Then, the play π^* matches π up to repetitions. Indeed, by removing occurrences of Player 1 vertices in $G_{T,\mathcal{G}}$ from π and projecting both plays on V , we obtain the same path. Recall that a Player 1 vertex

w^1 has the same parity index of its predecessor $w \in V_2$. It follows that since π satisfies Player 1's parity objective, so does π^* . In the second case, π reaches a sink $\langle s, T(s) \rangle$, where $s \in S$. Then π^* reaches a configuration $\langle s, B \rangle$. Since T satisfies the average property, we have $B \geq \text{fr}(s)$, and Player 1 wins \mathcal{G} . In the final case, π reaches a sink $\langle v, \top \rangle$. Let $\langle u, B_1 \rangle$ be the corresponding configuration in \mathcal{G} . Note that B_1 is strictly greater than $T(u) \oplus 0^*$. Let $B' < B_1$ that agrees with B_1 on the advantage and $\langle u, B' \rangle \in V_2$. Player 1 intuitively adds $B_1 - B'$ to his "spare change" account and plays as if his budget is B' by restarting $G_{T,\mathcal{G}}$ from $\langle u, B' \rangle$. Since the sum of budgets is fixed and restarting f^* in this manner strictly increases his spare change, it follows that this last case can only occur finitely often. Thus, eventually either of the first two cases must occur implying that f^* is winning in \mathcal{G} . ◀

► **Corollary 20.** *In the proof of Lem. 19, we construct a winning Player 1 strategy f^* . Note that f^* only keeps track of a vertex in $G_{T,\mathcal{G}}$. Thus, its memory size equals the size of $G_{T,\mathcal{G}}$, which is linear in the size of \mathcal{G} . This is significantly smaller than previously known constructions in parity and reachability bidding games, where the strategy size is polynomial in k , and is thus exponential when k is given in binary.*

The following lemma shows completeness; namely, that a correct guess of T implies that Player 1 wins from every vertex in $G_{T,\mathcal{G}}$.

► **Lemma 21.** *If $T = \text{Th}_{\mathcal{G}}$, then Player 1 wins from every vertex in $G_{T,\mathcal{G}}$.*

Proof. We describe the idea of the proof and the details can be found in the full version [10]. Assume towards contradiction that $T \equiv \text{FrPa-Th}$ and there is $\langle v, B \rangle \in V_2$ that is losing for Player 1. Since $T(v) \leq B$, Player 1 has a winning strategy f^* in \mathcal{G} starting from $\langle v, B \rangle$. Let g be a memoryless winning strategy for Player 2 in $G_{T,\mathcal{G}}$ starting from $\langle v, B \rangle$. Based on g , we construct a Player 2 strategy g^* in \mathcal{G} and show that it is winning against f^* , which contradicts our assumption that f^* is a Player 1's winning strategy. Note that since f^* is fixed, when g^* selects a bid, it is in response to the bid chosen by f^* . Let us consider an arbitrary Player 1 strategy f in $G_{T,\mathcal{G}}$, which by our earlier assumption is losing for Player 1 from vertex $\langle v, B \rangle$ of V_2 (because all Player 1 strategies are losing from there). Intuitively, we construct g^* such that it follows g as long as f^* follows f . By doing so, Player 2 maintains a similar correspondence between the play in \mathcal{G} and $G_{T,\mathcal{G}}$ as in the above: when \mathcal{G} is in configuration $\langle v, B_1 \rangle$, then $G_{T,\mathcal{G}}$ is in vertex $\langle v, B_1 \rangle$. Since the two plays traverse the same vertices in V and g is winning, if f^* always agrees with f , the resulting play will be winning for Player 2. Thus, f^* must not agree with f at some point. Either he bids differently from $b^T(v, B)$, or, upon winning he chooses a vertex u which is not in the allowed set of vertices. First assume that he bids b at $\langle u, B \rangle$. If $b > b(u, B)$, then g^* bids 0, intuitively causing Player 1 to pay too much for a bidding win. If $b < b^T(u, B)$, g^* bids $b^T(u, B)$ intuitively buying a win cheaply. In both cases, we show that \mathcal{G} reaches a configuration $c' = \langle u', B' \rangle$ for $B' < T(u)$. Since we assume $T \equiv \text{Th}_{\mathcal{G}}$, Player 2 has a winning strategy from c' , which she uses to win \mathcal{G} . Second, we show that when he chooses some vertex $u' \notin A(v)$ following a winning bid of $b^T(v, B)$, the game reaches a configuration $\langle u', B' \rangle$, such that $B' < T(u')$, and again Player 2 uses a winning strategy to win \mathcal{G} . ◀

Finally, we verify that $T \leq \text{Th}_{\mathcal{G}}$. We define a function $T' : V \rightarrow [k] \cup \{k+1\}$ as follows. For $v \in V$, when $T(v) > 0$ we define $T(v) = k^* \ominus (T(v) \ominus 0^*)$, and $T'(v) = k+1$ otherwise. Lem. 10 shows that T' satisfies the average property. We proceed as in the previous construction only from Player 2's perspective. We construct a partial strategy $f_{T'}$ for Player 2 from T' just as f_T is constructed from T , and construct a turn-based parity game $G_{T',\mathcal{G}}$. Let $\text{Th}_{\mathcal{G}}$

denote Player 2's threshold function in \mathcal{G} . That is, at a vertex $v \in V$, Player 2 wins when her budget is at least $\text{Th}_{\mathcal{G}}^2(v)$ and she loses when her budget is at most $\text{Th}_{\mathcal{G}}^2(v) \ominus 0^*$. Existence of $\text{Th}_{\mathcal{G}}^2$ follows from Thm. 17. Applying Lemmas 19 and 21 to Player 2, we obtain the following.

► **Lemma 22.** *If Player 2 wins from every vertex in $G_{T',\mathcal{G}}$, then $T' \geq \text{Th}_{\mathcal{G}}^2$. If $T' \equiv \text{Th}_{\mathcal{G}}^2$, then Player 2 wins from every vertex of $G_{T',\mathcal{G}}$.*

Given a frugal-parity discrete-bidding game $\mathcal{G} = \langle V, E, k, p, S, \mathbf{fr} \rangle$, a vertex $v \in V$, and $\ell \in [k]$, we guess $T : V \rightarrow [k] \cup \{k+1\}$ and verify that it satisfies the average property. Note that the size of T is polynomial in \mathcal{G} since it consists of $|V|$ numbers each of size $O(\log k)$. We construct $G_{T,\mathcal{G}}$ and $G_{T',\mathcal{G}}$, guess memoryless winning strategies for Player 1 and Player 2, respectively. We check whether $T(v) \geq \ell$, and answer accordingly. Correctness follows from Lemmas 19, 21, and 22. We thus obtain our main result.

► **Theorem 23.** *The problem of finding threshold budgets in frugal-parity discrete-bidding games is in NP and coNP.*

6 Discussion

We study, for the first time, the problem of computing threshold budgets in discrete-bidding games in which the budgets are given in binary. Previous algorithms for reachability and parity discrete-bidding games have exponential running time in this setting. We developed two algorithms for finding threshold budgets. The algorithms are complementary, and mirror the situation in the continuous setting; there too, there are two proof techniques to show results for threshold budgets, a fixed-point technique and an NP algorithm that relies on knowledge of the structure of threshold budgets. Prior to this work, a fixed-point algorithm was only known for reachability discrete-bidding games [14]. While our fixed-point algorithm for parity discrete-bidding games has exponential worst case running time, it sheds light on the structure of threshold budgets in these games. A structure that was crucial for our NP and coNP membership proof. This latter proof adds to the previously observed good news on discrete-bidding games: parity discrete-bidding games are not only a sub-class of concurrent games that is determined [1], we show that it is a sub-class of concurrent games that are represented in an exponentially-succinct manner and can still be solved in NP and coNP.

We leave open the exact complexity of finding threshold budgets. For the lower bound, it was shown in [1] that turn-based parity games reduce to parity discrete-bidding games with constant sum of budgets. Since solving turn-based parity games is a long-standing open problem, we expect that it will be challenging to find a polynomial-time algorithm for solving parity discrete-bidding games. Still, improved upper bounds might be possible to obtain. For example, a quasi-polynomial time algorithm for parity discrete-bidding games or a polynomial-time algorithm for reachability or Büchi discrete-bidding games.

References

- 1 M. Aghajohari, G. Avni, and T. A. Henzinger. Determinacy in discrete-bidding infinite-duration games. In *Proc. 30th CONCUR*, volume 140 of *LIPIcs*, pages 20:1–20:17, 2019.
- 2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
- 3 O. Amir, E. Kamar, A. Kolobov, and B. J. Grosz. Interactive teaching strategies for agent training. In *Proc. 25th IJCAI*, pages 804–811. IJCAI/AAAI Press, 2016.
- 4 N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts. *IACR Cryptology ePrint Archive*, 2016:1007, 2016.

- 5 G. Avni and T. A. Henzinger. A survey of bidding games on graphs. In *Proc. 31st CONCUR*, volume 171 of *LIPICs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 6 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. *J. ACM*, 66(4):31:1–31:29, 2019.
- 7 G. Avni, T. A. Henzinger, and R. Ibsen-Jensen. Infinite-duration poorman-bidding games. In *Proc. 14th WINE*, volume 11316 of *LNCS*, pages 21–36. Springer, 2018.
- 8 G. Avni, T. A. Henzinger, and Đ. Žikelić. Bidding mechanisms in graph games. In *In Proc. 44th MFCS*, volume 138 of *LIPICs*, pages 11:1–11:13, 2019.
- 9 G. Avni, I. Jecker, and Đ. Žikelić. Infinite-duration all-pay bidding games. In *Proc. 32nd SODA*, pages 617–636, 2021.
- 10 G. Avni and S. Sadhukhan. Computing threshold budgets in discrete-bidding games. *CoRR*, abs/2210.02773, 2022. [arXiv:2210.02773](https://arxiv.org/abs/2210.02773).
- 11 J. Bhatt and S. Payne. Bidding chess. *Math. Intelligencer*, 31:37–39, 2009.
- 12 K. Chatterjee, A. K. Goharshady, and Y. Velner. Quantitative analysis of smart contracts. In *Proc. 27th ESOP*, pages 739–767, 2018.
- 13 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.
- 14 M. Develin and S. Payne. Discrete bidding games. *The Electronic Journal of Combinatorics*, 17(1):R85, 2010.
- 15 O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th STOC*, pages 224–233. ACM, 1998.
- 16 A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial games under auction play. *Games and Economic Behavior*, 27(2):229–264, 1999.
- 17 A. J. Lazarus, D. E. Loeb, J. G. Propp, and D. Ullman. Richman games. *Games of No Chance*, 29:439–449, 1996.
- 18 R. Meir, G. Kalai, and M. Tennenholtz. Bidding games and efficient allocations. *Games and Economic Behavior*, 112:166–193, 2018. [doi:10.1016/j.geb.2018.08.005](https://doi.org/10.1016/j.geb.2018.08.005).
- 19 S. Muthukrishnan. Ad exchanges: Research issues. In *Proc. 5th WINE*, pages 1–12, 2009.
- 20 Y. Peres, O. Schramm, S. Sheffield, and D. B. Wilson. Tug-of-war and the infinity laplacian. *J. Amer. Math. Soc.*, 22:167–210, 2009.
- 21 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
- 22 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. [doi:10.1016/S0304-3975\(98\)00009-7](https://doi.org/10.1016/S0304-3975(98)00009-7).

A Example of the construction of $G_{T,\mathcal{G}}$

► **Example 1.** We apply the construction to the frugal-reachability discrete-bidding game \mathcal{G}_1 depicted in Fig. 1. We use the two functions $T_1, T_2 : V \rightarrow V \rightarrow [k] \cup \{k+1\}$ that are specified in the figure, both of which satisfy the average property and correspond to the vectors $T_1 = \langle 4, 3^*, 3, 2 \rangle$ and $T_2 = \langle 5, 4^*, 3^*, 2 \rangle$.

The turn-based game G_{T_1, \mathcal{G}_1} is depicted in Fig. 2a and G_{T_2, \mathcal{G}_1} in Fig. 2b. In these two games, the allowed actions given by f_{T_1} and f_{T_2} from each vertex are singletons. Thus, Player 1 has no choice of successor vertex when winning a bidding, and so we omit Player 1 vertices from the figure. That is, all vertices are controlled by Player 2. Since the games are reachability games, we omit the parity indices from the vertices. Player 1’s goal in both games is to reach a sink. We label each edge with the bids of the two players that it represents. Each vertex c has two outgoing edges labeled by $\langle b_1, 0 \rangle$ and $\langle b_1, b_1 \oplus 0^* \rangle$, where b_1 is the bid that f_{T_1} or f_{T_2} prescribes at c . There are exceptions like $\langle v_1, 5 \rangle$ in G_{T_2, \mathcal{G}_1} where $b_1 = 1$ and Player 2 cannot bid $b_1 \oplus 0^* = 1^*$ since it exceeds her available budget when $k = 5$.

30:18 Computing Threshold Budgets in Discrete-Bidding Games

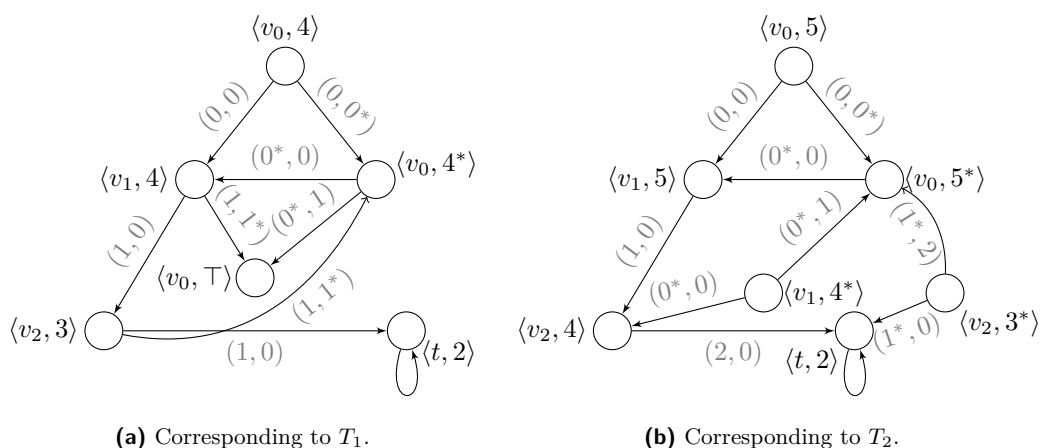


Figure 2 Turn-based games corresponding to G_1 and two different guesses.

Note that in G_{T_1, G_1} has a cycle. Thus, Player 1 does not win from every vertex and T_1 does not coincide with the threshold budgets. On the other hand, G_{T_2, G_1} is a DAG. Thus, no matter how Player 2 plays, Player 1 wins from all vertices, which means that $T_2 = \text{Th}_{G_1}$. ◀

Dependency Matrices for Multiplayer Strategic Dependencies

Dylan Bellier ✉ 

Univ Rennes, IRISA, CNRS, France

Sophie Pinchinat ✉

Univ Rennes, IRISA, CNRS, France

François Schwarzentruber ✉

Univ Rennes, IRISA, CNRS, France

Abstract

In multi-player games, players take their decisions on the basis of their knowledge about what other players have done, or currently do, or even, in some cases, will do. An ability to reason in games with temporal dependencies between players' decisions is a challenging topic, in particular because it involves imperfect information. In this work, we propose a theoretical framework based on *dependency matrices* that includes many instances of strategic dependencies in multi-player imperfect information games. For our framework to be well-defined, we get inspiration from quantified linear-time logic where each player has to label the timeline with truth values of the propositional variable she owns. We study the problem of the existence of a winning strategy for a coalition of players, show it is undecidable in general, and exhibit an interesting subclass of dependency matrices that makes the problem decidable: the class of perfect-information dependency matrices.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects

Keywords and phrases Temporal dependency, Delay games, Strategic reasoning, Temporal logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.31

Supplementary Material

Interactive Resource: <https://francoisschwarzentruber.github.io/fsttcs2022/>

Software (Source Code): <https://github.com/francoisschwarzentruber/fsttcs2022>

archived at `swh:1:dir:eed0f57a83b2979abe84fd80bd804a6d730539b3`

1 Introduction

In perfect-information multi-player games, decisions of players depend on their knowledge about what other players have done so far. This setting is adopted in various logics for strategic reasoning such as Alternating-time Temporal Logics [2] and Strategy Logic [4]. A *strategy* for a player assigns to each history the next action to play. In imperfect-information games, a player may have partial knowledge about what other players did, enforcing her strategy to depend only on her knowledge about the current history.

However, there are situations where dependencies involve the knowledge about the future of a play. For instance, Grove and Clarkson in [7] developed an online algorithm for the bin packing problem with a *look-ahead* that takes advantage of information on some future items. Look-ahead also exists in parsing: for LL(1) (*for Left to right, Leftmost derivation*) grammars, the production rule to be chosen depends on the next symbol in the read word [1]. The extreme case occurs when the decisions depend on the *entire* future of the play. This is the case for *offline* algorithms, where the future is finite (for bin packing it is given by the sequence of all incoming items). It also appears for an infinite future, such as in Quantified Propositional Linear-time Temporal Logic (QPTL) [14]: for a formula $\forall a \exists b \varphi$, the choice of the truth values of proposition *b* depends on all those of proposition *a*.



© Dylan Bellier, Sophie Pinchinat, and François Schwarzentruber;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 31; pp. 31:1–31:21



Leibniz International Proceedings in Informatics

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

In addition, players may experience delays to receive information. For instance, a proponent may have to play without having received yet information about the last three moves of her opponent. Klein et al. in [8] formalized *delay games* as an extension of Gale-Stewart games [6], i.e. two player infinite games with perfect information.

Our paper proposes a unifying theoretical framework to specify dependencies. To this aim, we define the notion of a *dependency matrix* D . The entry $D[a, b]$ is a generalized integer (i.e. in $\mathbb{Z} \cup \{-\infty, +\infty\}$) so that

“Player a ’s decision at time-step $t_{current}$ depends on Player b ’s decisions up to time-step $t_{current} + D[a, b]$ ”.

The semantics of a dependency matrix relies on an involved machinery based on an imperfect-information multi-player game, called the *meta game*. The positions of the game, called *configurations*, are all the possible partial labelings of the timeline by the players – and are thus in infinite number. The dynamics of the meta game is involved because the dependencies may desynchronize players in their choices for labeling a time point.

Moreover, some matrices encode circular dependencies between players, leading to deadlocked situations where none of the players can progress anymore in the meta game, preventing them from completing their labeling of the timeline. Upon the study of this phenomenon, we introduce the class of *progressing* matrices, that guarantee that any play in the meta arena provides a full labeling of the timeline for each propositional variable. We establish an effective property that characterizes these matrices. Thus, plays can be qualified as winning or losing according to some linear-time formula, here an LTL formula.

We then study the problem called EWS (Existence of Winning Strategies) of deciding, given a dependency matrix, a coalition (subset of players) and an LTL formula, the existence of a joint strategy for the coalition such that any play brought about by this strategy satisfies the LTL formula. Importantly, the imperfect-information feature of the meta game addresses two issues: first, this game is not determined in general, and second, winning strategies for the coalition need being *uniform*, a non-trivial notion in our rich setting since players may be desynchronized.

Although we prove that EWS is unsurprisingly undecidable, we exhibit the subclass of so-called *perfect information* dependency matrices for which EWS turns to a decidable problem. We first consider the perfect information property for matrices whose values range over \mathbb{Z} and show how EWS can be reduced to solving a two-player perfect-information parity game, yielding a 2-EXPTIME-complete complexity. We then generalize the perfect-information property to arbitrary matrices and provide a decision procedure for EWS that generalizes the one for QPTL [14], thus a non-elementary complexity.

To our knowledge, our proposal offers the first framework amenable for merging many, and yet remote, game settings such as concurrent or turn-based games [2], (two-player) delay games [8, 9, 16], logic QPTL [14], and Church Synthesis Problem (see the survey [5]) – it can be shown that our framework also subsumes DQBF (Dependency Quantified Boolean Formulae) [10].

Outline. In Section 2, we define dependency matrices, and show that they embed several settings of games. Section 3 contains the necessary background. In Section 4, we present the formal machinery to define an arena specified by a dependency matrix. In Section 5, we address the problem EWS of the existence of winning strategies and show its undecidability in the general case. Next, in Section 6, we design the decision procedure for EWS when restricted to a perfect-information matrix input. We conclude our contribution in Section 7.

2 Dependency Matrices and Examples

In this section, we propose the generic notion of dependency matrix and show that it subsumes several classic settings in games. We denote players by lowercase letters such as a, b, c , etc. A dependency matrix specifies the mutual dependencies between players' decisions in a game where each player owns an atomic proposition and aims at filling the whole timeline with a valuation for it at each time point. Formally,

► **Definition 1.** A dependency matrix (or simply a matrix) over a finite set \mathcal{P} of at least two players is a matrix $D = (D[a, b])_{a \neq b \in \mathcal{P}}$, and whose values range over $\mathbb{Z} \cup \{-\infty, +\infty\}$.

In a dependency matrix $(D[a, b])_{a \neq b \in \mathcal{P}}$ over \mathcal{P} , the value $D[a, b]$ describes how Player a 's decisions depends on Player b 's: Player a 's decision for choosing the valuation at time point t depends on the ones chosen by Player b at all time points in the interval $[0, t + D[a, b]]$. As such, whenever $D[a, b] < 0$, Player a 's decision at time point t is independent of the decisions made by Player b up to some time point before t . In particular, if $D[a, b] = -\infty$, Player a 's decisions is independent of any of Player b 's.

On the contrary, when $D[a, b] \geq 0$, Player a 's decision at t depends on some Player b 's decisions up to some time point after t , so that Player a is not able to make her decision without this required information. In particular, if $D[a, b] = +\infty$, Player a 's decisions do depend on the decisions of Player b 's over all the timeline.

Because it is natural to consider that a player is aware of her own decisions so far, values on the diagonal $D[a, a]$ are irrelevant and are left undefined. As such, the matrix line $D[a, \cdot]$ specifies the dependencies of Player a with respect to all other players, and her ability to make a decision is constrained by all these dependencies. Unsurprisingly, some matrices may yield blocking situations for some players, an issue that we address in the sequel.

Beforehand, we illustrate how several settings in games can be captured with matrices.

► **Example 2 (Concurrent Game).** In a standard concurrent game (as in logics ATL , ATL^*), players have to concurrently choose a move. Thus the move of one player can only depend on the strict past of the history of moves. The corresponding matrix for 3 players is D_1 , where strict past is reflected by the value -1 .

$$D_1 = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{pmatrix} \cdot & -1 & -1 \\ -1 & \cdot & -1 \\ -1 & -1 & \cdot \end{pmatrix} \end{matrix}$$

► **Example 3 (Round Robin Game).** In a Round Robin game, players play in turn: first Player a , then b , then c . The matrix is D_2 : decisions of Player a depend on the strict past, hence the values -1 in the a -row; decisions of b depend on the non-strict past of a (hence the value 0), and the strict past of c (hence the value -1); decisions of c depend on the non-strict past of the other players.

$$D_2 = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{pmatrix} \cdot & -1 & -1 \\ 0 & \cdot & -1 \\ 0 & 0 & \cdot \end{pmatrix} \end{matrix}$$

31:4 Dependency Matrices for Multiplayer Strategic Dependencies

► **Example 4 (QPTL).** In QPTL, dependencies stem from the order of the quantifiers: in the formula $\exists a \forall b \exists c \varphi$ where φ is an LTL-formula, Player a plays first on the full timeline and independently of the others. Then b only depends on what Player a did. Finally, c depends on what both Players a and b did. All this is reflected by matrix D_3 .

$$D_3 = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{pmatrix} \cdot & -\infty & -\infty \\ +\infty & \cdot & -\infty \\ +\infty & +\infty & \cdot \end{pmatrix} \end{matrix}$$

► **Example 5 (Church Synthesis).** The Church Synthesis problem (see the survey [5]) consists in responding to a stream of inputs by a stream of outputs, so that a given property holds. If Player a and Player b are in charge of the output, Player c and Player d are in charge of the input, the players dependencies are captured by matrix D_4 : output Players a and b only depend on the past values of the input players, and that input Players c and d see all values and have to respond on the spot.

$$D_4 = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} \cdot & -1 & -1 & -1 \\ -1 & \cdot & -1 & -1 \\ 0 & 0 & \cdot & -1 \\ 0 & 0 & -1 & \cdot \end{pmatrix} \end{matrix}$$

► **Example 6 (Fixed Delay games).** A delay game is a two-player game, say between Player a and Player b . Player a must make a given finite number k of moves beforehand and then, Players a and b play in a turn based manner, hence maintaining the delay between them (see Klein et al. [8]).¹ This setting is represented by the matrix D_5 .

$$D_5 = \begin{matrix} & \begin{matrix} a & b \end{matrix} \\ \begin{matrix} a \\ b \end{matrix} & \begin{pmatrix} \cdot & -k \\ k-1 & \cdot \end{pmatrix} \end{matrix}$$

In the next section, we fix some notations and recall some useful definitions to develop our theory around matrices.

3 Background

Given a finite alphabet Σ , we use the standard notations Σ^* , Σ^ω and Σ^∞ for the set of finite words, infinite words and their union respectively, and ε to denote the empty word. Given two words $u, w \in \Sigma^*$, we write $u \cdot w$ for their concatenation. Given a non-empty word $u = u_0 u_1 \cdots u_n$, we let $|u| := n + 1$ be its length and set $|\varepsilon| = 0$. For $k \leq n$, we write $u[k]$ for the letter u_k , $u[:k]$ for the k -th prefix $u_0 \cdots u_k$ and $u[k:]$ for the k -th suffix $u_k \cdots u_n$.

We now recall the basics of *Linear-time Temporal Logic* (LTL) [13]. An LTL formula φ (over a set AP of propositions) is evaluated on a labeling of the (discrete) timeline \mathbb{N} , called an *LTL assignment*² $\lambda \in (\{\top, \perp\}^\omega)^{\text{AP}}$. We classically write $\lambda \models \varphi$ whenever λ satisfies

¹ In their setting, the delay is defined with a delay function that gives at each round the number of moves the input Player has to make. However, Klein et al. mainly studied the fixed delay setting where the function is set to 1 after the first round.

² also named *trace* in the model-checking setting.

φ (we omit the semantics here and refer to [13]). Alternatively, an LTL assignment can be seen as an infinite word over the alphabet $\{\top, \perp\}^{\text{AP}}$, which makes a tight connection between logic and automata: given an LTL formula φ , one can build – via the Vardi-Wolper construction [17] together with the Safra-like translation from Büchi to parity acceptance condition [11] – a deterministic *infinite-word parity automaton* \mathcal{A}_φ whose language $\mathcal{L}(\mathcal{A}_\varphi)$ is composed of all LTL assignments that satisfy φ .

Noticeably, LTL can be extended with propositional quantifications to *Quantified Propositional Temporal Logic* (QPTL), which enjoys the prenex normal form [15]. Therefore, we can assume that QPTL formulas are all of the form $\vec{Q} \varphi$, where \vec{Q} is a finite sequence of propositional quantifications ($\exists a$ or $\forall a$, where $a \in \text{AP}$) and φ is an LTL formula.

In this paper, we consider vectors of words over the alphabet $\{\top, \perp\}$, indexed by a finite subset \mathcal{P} of AP. Given a vector $U = (U_a)_{a \in \mathcal{P}}$ of words over $\{\top, \perp\}$ and a player $b \in \mathcal{P}$, we denote by $U(b)$ the b 's component of U . We extend all notations for words to word vectors with their meaning component-wise. In addition, we introduce the notation $U +_a u$ for the word vector obtain by concatenating the word u to $U(a)$.

As we will see, in our framework, the set AP of propositions for logics LTL and QPTL will be the breeding ground to pick the finite set \mathcal{P} of players for our matrices.

4 The Formal Setting of Dependency Matrices

From a matrix D over \mathcal{P} , a coalition $\Gamma \subseteq \mathcal{P}$ and an LTL formula φ , we derive the *meta game* $\langle D, \Gamma, \varphi \rangle$ that specifies how players can progress to label the timeline with truth values for their proposition, what the coalition is, and what the winning condition is. Precisely, the goal of the coalition is to make φ true.

The arena for the meta game, fully determined by the matrix D , is a modified multiplayer Gale-Stewart arena [6] taking into account the information flow between players according to D . We distinguish *progressing* matrices that yield arenas with only infinite plays where no player is blocked, thus resulting in an LTL assignment. We also provide a graph-based polynomial-time algorithm to characterize progressing matrices.

Sticking to progressing matrices, we develop the proper notions of (player and coalition) strategies, along with the property of *uniformity* of a strategy that takes into account the imperfect information feature of our games. Intuitively, a strategy in the meta game is uniform if it depends only on the informations prescribed by the matrix. For pedagogical purposes, we start with bounded-value matrices, then consider arbitrary ones.

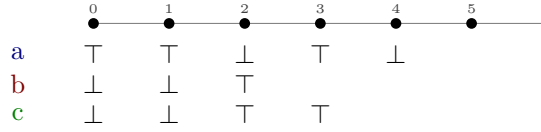
4.1 The Meta Arena of a Matrix

We fix a matrix D with values in \mathbb{Z} and we describe the *meta arena* of D . A position in the meta arena is called a *configuration*, that is a word vector C that reflects the labeling over $\{\top, \perp\}$ chosen so far by each player of \mathcal{P} . The set of configurations is denoted by $\mathcal{C} := (\{\top, \perp\}^*)^{\mathcal{P}}$ and the initial configuration C^0 is the empty vector, namely $\varepsilon^{\mathcal{P}}$.

► **Example 7.** Figure 1 shows the configuration C in which Player a played the word $C(a) = \top\top\perp\top\perp$, Player b played $C(b) = \perp\perp\top$, and Player c played $C(c) = \perp\perp\top\top$.

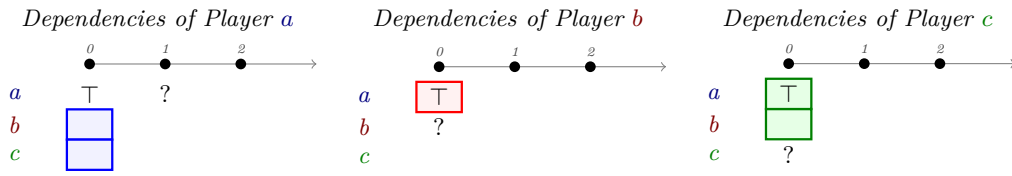
We define the dynamics of the game, namely which player can play/*progress*, in a given configuration and which moves are available to her. Here is an intuitive example of those dynamics with a Round Robin matrix.

31:6 Dependency Matrices for Multiplayer Strategic Dependencies



■ **Figure 1** A configuration C where Player a has chosen her labeling up to time point 4, Player b up to time point 2 and Player c up to 3.

► **Example 8.** For the matrix D_2 of Example 3 and for the first round, only Player a can make a move of length 1. Then, assuming Player a chooses \top , the dependencies for each player are depicted separately below: in each picture, the squares identify expected information for the considered player to make her decision about the question mark (?).



Observe that, here, neither Player a , nor Player c can make a move as the labeling of Player b at time point 0 is not set. On the contrary, Player b is able to progress.

A move of a Player a is a word $u_a \in \{\top, \perp\}^*$ that is to extend her labeling along the timeline. For Player a to progress in configuration C , all her dependencies must be fulfilled. This is formalized as follows: in order to make a move (necessarily starting at the time point $t = |C(a)|$), Player a needs to access the labeling of Player b up to time point $t + D[a, b]$, included. Therefore, the value $\alpha_{a,b}^C := |C(b)| - (D[a, b] + |C(a)|)$ characterizes the length of a move available to Player a with regard to her dependency on Player b only. Thus, the overall *progress value* of Player a , written α_a^C , takes into account all quantities $\alpha_{a,b}^C$ for $b \neq a$ in a conjunctive manner, leading to consider the most restrictive one. Formally:

$$\alpha_a^C \stackrel{\text{def}}{=} \max(0, \min_{b \neq a}(\alpha_{a,b}^C)) \quad (1)$$

As such, α_a^C is the maximum number of steps that Player a can perform in configuration C . Note that, if some $\alpha_{a,b}^C$ is negative (including 0), then Player a is stuck in C because Player b has not yet provided the expected information.

Based on the progress value, a *legitimate move* for Player a in a configuration C is a word in $\{\top, \perp\}^{\alpha_a^C}$. Then, a *legitimate joint move* in C is a vector of words $u \in (\{\top, \perp\}^*)^{\mathcal{P}}$ such that, for every Player a , $u(a)$ is a legitimate move for a . We can now define the *move function* between configurations:

$$\Delta : \mathcal{C} \times (\{\top, \perp\}^*)^{\mathcal{P}} \rightarrow \mathcal{C}$$

$$(C, u) \mapsto \begin{cases} C \cdot u & \text{when } u \text{ is a legitimate joint move in } C \\ \text{undefined} & \text{otherwise} \end{cases}$$

with u a legitimate joint move. Remark that all players that can move have to play concurrently and greedily (i.e. their maximum number α_a^C of actions). A configuration C is said *reachable* if there is a finite sequence of legitimate joint moves that leads to C from the initial configuration C^0 .

► **Proposition 9.** Every reachable configuration, has a unique predecessor by Δ .

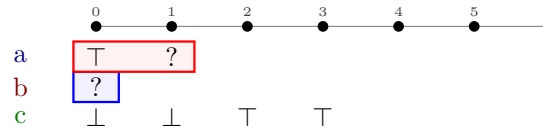
Proof. Suppose that there are two reachable configurations C^1 and C^2 and two legitimate joint moves u^1 and u^2 such that $C^1 \cdot u^1 = C^2 \cdot u^2$. Then, consider a reachable configuration C' that is a prefix of both C^1 and C^2 . Suppose toward contradiction that there are legitimate joint moves $u^{1'} \neq u^{2'}$ with $C^{1'} = C' \cdot u^{1'}$ prefix of C^1 and $C^{2'} = C' \cdot u^{2'}$ prefix of C^2 . Since $u^{1'} \neq u^{2'}$, there is $t \in \mathbb{N}$ and $a \in \mathcal{P}$ such that $C^{1'}(a)[t] \neq C^{2'}(a)[t]$. By definition, $C^{i'}$ is a prefix of C^i for $i \in \{1, 2\}$. Hence $C^1(a)[t] \neq C^2(a)[t]$ which is in contradiction with $C^1 \cdot u^1 = C^2 \cdot u^2$. In conclusion, if $C^1 \cdot u^1 = C^2 \cdot u^2$, then $C^1 = C^2$ and $u^1 = u^2$. ◀

By Proposition 9, the meta arena of reachable configurations is a tree and every reachable configuration contains all moves since the start of the game.

Observe that some meta arenas have reachable configurations C where for every Player a , we have $\alpha_a^C = 0$. This happens because of cyclic dependencies. Here is an example.

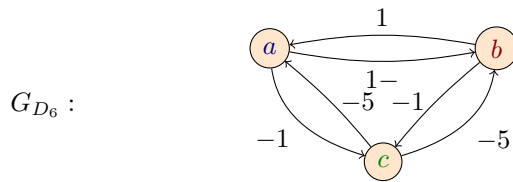
► **Example 10.** Consider the matrix D_6 on the right. Observe that Player b cannot make any move because she needs Player a 's labeling at time point 1, but for Player a to label time point 1, she needs the label of Player b at time point 0. This deadlocked situation is depicted in Figure 2. However, observe that Player c can progress independently up to time point 3.

$$D_6 = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{pmatrix} \cdot & -1 & -1 \\ +1 & \cdot & -1 \\ -4 & -4 & \cdot \end{pmatrix} \end{matrix}$$



■ **Figure 2** Player a wants to know the moves of Player b and reciprocally.

Situations where some players eventually get stuck can be characterized by analyzing some graph: for the case of Example 10, the graph is depicted below, and interestingly, it contains the cycle (a, b, a) whose weight is 0, a positive value. We will see in the next section, where the graph is formally defined, that such a cycle provides evidence that Players a and b eventually get stuck.



4.2 Progressing Matrix

We aim at characterizing matrices where no player gets stuck because of cyclic dependencies, so that each play yields an assignment of the timeline in order to interpret the LTL winning condition. Such matrices are called *progressing* and can be identified by means of their adjacency weighted graph, here called the *dependency graph*.

► **Definition 11.** Given a matrix $D = (D[a, b])_{a \neq b \in \mathcal{P}}$, the dependency graph of D is the weighted directed graph $G_D = (V, E, r)$ where:

- $V = P$ is the set of vertices,
- $E = \{(a, b) \mid a \neq b\}$ is the set of edges,
- $r(a, b) = D[a, b]$ is the weight of the edge (a, b) .

The following proposition gives a characterization of progressing matrices.

► **Proposition 12.** *A matrix D is progressing if, and only if, its dependency graph G_D has no non-negative-weighted cycle.*

The graph G_{D_6} of Example 10 has a non-negative 0-weight cycle (a, b, a) , so, matrix D_6 is not progressing. As a corollary, we have the following:

► **Theorem 13.** *Deciding if a matrix is progressing is in PTIME.*

Proof. The size of the dependency graph is linear in the size of the matrix and finding a non-negative cycle is polynomial in the size of the graph. ◀

From now on, unless stated otherwise, we only consider progressing matrices, that we keep calling “matrices” for simplicity. On the basis of such matrices, only infinite-horizon plays take place that consist in consecutive applications of the move function Δ in the meta arena (see page 6):

► **Definition 14.** *A play in the meta arena associated to a matrix is an infinite sequence of configurations $(C^n)_{n \in \mathbb{N}}$ where C^0 is the empty configuration, and for every $n \in \mathbb{N}$, C^{n+1} is the successor of C^n .*

The interested reader can explore the dynamics of plays at:

<https://francoisschwarzentruber.github.io/fsttcs2022>

Notice that along a play $(C^n)_{n \in \mathbb{N}}$, C^{n+1} extends C^n , so that to the limit, the play naturally yields a temporal assignment λ of the timeline: for every $t \in \mathbb{N}$, and every Player a , we let $\lambda(t)(a) \stackrel{\text{def}}{=} C^n(a)(t)$, for a sufficiently large integer n so that $C^n(a)(t)$ is defined.

The next section focuses on strategies and winning strategies in the meta arena, where we discuss how a strategy complies with a matrix.

4.3 Strategies in the Meta Arena

In this section, we fix a matrix D over \mathcal{P} , a coalition $\Gamma \subseteq \mathcal{P}$ and an LTL formula φ . Classically, a strategy maps histories to moves. However, since in our setting, a configuration fully characterizes a history (Proposition 9), we can equivalently define strategies as mappings from configurations to moves. A *strategy* for Player $a \in \Gamma$ is a function $f : \mathcal{C} \rightarrow \{\top, \perp\}^*$ where $f(C)$ prescribes a legitimate move for Player a .

Furthermore, we define a *joint strategy* for the Γ as a function $F : \mathcal{C} \rightarrow (\{\top, \perp\}^*)^\Gamma$ such that $F(C)(a)$ is a legitimate move for Player $a \in \Gamma$. A joint strategy F provides a strategy F_a for each Player $a \in \Gamma$ defined by: $F_a(C) \stackrel{\text{def}}{=} F(C)(a)$.

Given a joint strategy F for a coalition Γ , a play $(C^n)_{n \in \mathbb{N}}$ is an *outcome* of F if for any $n \in \mathbb{N}$ and any Player $a \in \Gamma$, we have $C^{n+1}(a) = C^n(a) \cdot F_a(C^n)$. We denote by $out(F)$ the set of outcomes of F . A joint strategy F is *winning* φ whenever all assignments associated to the plays in $out(F)$ satisfy φ .

However, in our dependency-based setting, a strategy is relevant only if it is *uniform*, in the sense that they only rely on the information available to the player. We illustrate this important feature in Example 15.

► **Example 15.** *Consider matrix D_7 and configurations C^1 and C^2 below.*

$$D_7 = \begin{matrix} & a & b \\ \begin{matrix} a \\ b \end{matrix} & \begin{pmatrix} \cdot & -2 \\ -2 & \cdot \end{pmatrix} \end{matrix} \quad C^1 = \begin{matrix} & 0 & 1 \\ & \bullet & \bullet \\ a & \top & \top \\ b & \perp & \perp \end{matrix} \quad C^2 = \begin{matrix} & 0 & 1 \\ & \bullet & \bullet \\ a & \top & \top \\ b & \perp & \top \end{matrix}$$

When Player a in C^1 comes to label time point 2, and since at time point 2, she can only access Player b 's labeling up to time point 0, she cannot distinguish it from C^2 . However, once she labels time point 2, she is able to access Player b 's label at time point 1, and is allowed to take this information into account before choosing her label at time point 3. Now, according to the matrix D_7 , we have $\alpha_a^{C^1} = \alpha_a^{C^2} = 2$. Although Player a cannot distinguish between C^1 and C^2 , she is allowed to choose different 2-length moves that only differ in their second letters. Indeed, her choice at time point 2 has to be uniform (and therefore the same in both configurations C^1 and C^2). On the contrary, she may play differently for her choice at time point 3. For instance, the overall move in C^1 can be $\top\perp$ while it is $\top\top$ in C^2 .

We formalize the phenomenon described in Example 15 with equivalence relations between configurations. In the example, Player a has to choose a move u_0u_1 but C^1 and C^2 are indistinguishable for her when it comes to choosing the first letter u_0 . However, they can be distinguished for the choice of the second letter u_1 . Then, we need multiple relations, one for each letter of a move.

Formally, we introduce an equivalence relation between configurations parameterized by a scope k : two configurations C^1 and C^2 are k -indistinguishable for Player a , denoted $C^1 \stackrel{a}{\sim}_D C^2$, whenever $|C^1(a)| = |C^2(a)|$, and for every Player $b \neq a$ and every $t \leq |C^1(a)| + D[a, b] + k$, we have:

- either both $C^1(b)(t)$ and $C^2(b)(t)$ are undefined (meaning k is greater than the progress of b in both configurations), or
- $C^1(b)(t) = C^2(b)(t)$.

We resort to relations $\stackrel{a}{\sim}_D^k$ to formalize the notion of uniform strategies in our framework, where the parameter k is meant to range over $[0, \min(\alpha_a^{C^1}, \alpha_a^{C^2})[$.

► **Definition 16.** A strategy f for Player a is D -uniform whenever for any two configurations C^1 and C^2 , and any natural number $k \leq \min(\alpha_a^{C^1}, \alpha_a^{C^2})$,

$$C^1 \stackrel{a}{\sim}_D^k C^2 \text{ implies } f(C^1)[:k] = f(C^2)[:k].$$

Observe that $C^1 \stackrel{a}{\sim}_D^{k+1} C^2$ implies $C^1 \stackrel{a}{\sim}_D^k C^2$. We generalize Definition 16 to joint strategies in a natural way by requiring the uniform property for every individual strategy of the joint strategy. For the rest of the paper, all strategies are implicitly D -uniform.

Before addressing the central decision problem of the existence of a winning joint strategy, we extend our setting to allow matrices with infinite values.

4.4 Arbitrary Matrices

Recall, by Definition 1, that a value $D[a, b] = -\infty$ indicates that Player a 's decision are independent from Player b 's. In particular, if the matrix line $D[a, \cdot]$ is all filled with value $-\infty$, Player a fills the whole timeline in the first round. On the contrary, $D[a, b] = +\infty$ forces Player a to wait until Player b has entirely filled the timeline.

A typical example of a matrix with infinite values is provided by the matrix D_3 of Example 4, and is reminiscent of what is expressed in the setting of the logic QPTL: in this example, a play takes place as follows. First, Player a chooses an a -assignment of the

31:10 Dependency Matrices for Multiplayer Strategic Dependencies

timeline. Second, since Player a is done and Player b is independent from Player c , Player b has all the required information for choosing the b -assignment over the timeline. Third and finally, Player c can proceed for the c -assignment, and the play ends.

For the cases that mix finite and infinite value, we consider first Example 17

► **Example 17.** Consider matrix D_8 below. In a play, Player b and Player c 's mutual dependencies enforce them to proceed in turn for choosing their respective labeling, while Player a cannot play until the other two have labeled the whole timeline.

$$D_8 = \begin{array}{c} \begin{array}{ccc} & a & b & c \\ a & \cdot & +\infty & +\infty \\ b & -\infty & \cdot & -1 \\ c & -\infty & 0 & \cdot \end{array} \end{array}$$

As observed in Example 17, $(\mathbb{Z} \cup \{-\infty, +\infty\})$ -valued matrices may yield to “transfinite” configurations (i.e. with possibly components in $\{\top, \perp\}^\omega$ instead of $\{\top, \perp\}^*$). As a result, a play may now be a finite sequence of (possibly infinite) sequences of configurations. We can adapt the notion of reachable configuration accordingly – since this is routine, we omit the precise definition here.

From now on, unless stated otherwise, we consider arbitrary matrices. In the next section, we address the decision problem EWS of the existence of a winning strategy.

5 Undecidability of EWS

We consider the following central decision problem EWS of deciding the Existence of a Winning (uniform) Strategy for a coalition of players (EWS for short):

► **Theorem 18.** Let EWS be the following problem:

Input: A matrix D , a coalition Γ and an LTL-formula φ .

Output: “Yes” if and only if there is a D -uniform joint strategy for Γ that is winning for φ . EWS is undecidable.

This result is unsurprising, given the imperfect-information nature of our multi-player game.

Theorem 18 is proved by reducing the Tiling problem [3] to EWS. Recall that the Tiling problem takes in input a finite set of square tiles and two binary connectivity relations over the tiles, that specify which pairs of tiles may be adjacent (resp. horizontally and vertically). The output is the answer to the question whether there exists a tiling of the plane, that is a mapping from \mathbb{N}^2 to the set of tiles such that any two of adjacent tiles respect the connectivity constraints. In a nutshell, our reduction involves four players τ_1, τ_2 (tilers) and c_1, c_2 (challengers): in a round, Challenger c_1 chooses a place (x, y) in \mathbb{N}^2 and privately communicates it to his tiler companion τ_1 by playing $\top^x \perp \top^y \perp^\omega$. Tiler τ_1 then responds by choosing a tile t independently of the choice of Challenger c_2 and plays $\top^t \perp^\omega$, and symmetrically for Players τ_2 and c_2 . This way, the two tilers play independently. The two different Challengers are used to test the binary relations by choosing adjacent places. The following matrix encodes this situation.

$$D_{\text{Tiling}} := \begin{array}{c} \begin{array}{cccc} & c_1 & c_2 & \tau_1 & \tau_2 \\ c_1 & \cdot & -\infty & -\infty & -\infty \\ c_2 & -\infty & \cdot & -\infty & -\infty \\ \tau_1 & +\infty & -\infty & \cdot & -\infty \\ \tau_2 & -\infty & +\infty & -\infty & \cdot \end{array} \end{array}$$

In the next section, we present a decidable sub-case.

6 Decidability of EWS for perfect-information Matrices

A close inspection of the proof of Theorem 18 reveals that not being able to circumvent the amount of information hidden to players is a matter. We introduce the subclass of *perfect-information* matrices where every player always has full information about the current configuration before proceeding, yielding a meta game that is turn-based with perfect information.

6.1 Definition and Properties

A meta game is perfect-information as long as two reachable configurations are not k -indistinguishable, for every k . Actually, not being 0-indistinguishable is sufficient, since k -indistinguishability are nested (see Section 4.3). Furthermore, for the meta arena to be turn-based, we must guarantee that in each round, only one player can progress. This yields the following definition.

► **Definition 19.** A matrix D is perfect-information if for every reachable configuration C :

- for every player a and reachable configuration $C' \neq C$, we have $C \not\sim_D^0 C'$ and
- there is exactly one player a such that $\alpha_a^C \geq 1$.

Remark that, by Definition 19, every move of a single player, from a reachable configuration that is not the initial one, is necessarily of length 1. Indeed, if a player could make a move of length strictly greater than 1, we could create another reachable configuration that would be 0-indistinguishable from the first one, a contradiction.

► **Lemma 20.** Let D be a perfect-information matrix, then $\alpha_a^C \leq 1$ for any non-initial reachable configuration C and every Player a .

Proof. By contradiction, suppose that there is a non-initial reachable configuration C^1 with $\alpha_a^{C^1} \geq 2$ for some Player a . Because D is progressing and perfect-information, there is a unique Player b that can progress in $\Delta^{-1}(C^1)$. If $b = a$, we would not have $\alpha_a^{C^1} \geq 2$ because players play greedily. Then, we have $a \neq b$.

We now exhibit a reachable configuration $C^2 \neq C^1$ such that $C^1 \sim_D^0 C^2$. Let u^1 be the joint move leading to C^1 , that is $\Delta(\Delta^{-1}(C^1), u^1) = C^1$. As exactly one player moves, $u^1(c) = \varepsilon$ for every Player $c \neq b$. We define the joint move u^2 as follows: for every Player $c \neq b$, let $u^2(c) = \varepsilon$ and $u^2(b) = \text{fliplast}(u^1(b))$ where *fliplast* flips the last letter of the word (mapping \top to \perp , and \perp to \top). For $C^2 = \Delta(\Delta^{-1}(C^1), u^2)$, we have $|C^1(a)| = |C^2(a)|$, and $C^1(c) = C^2(c)$ for Player $c \neq b$. We have $\alpha_{a,b}^{C^i} \geq 2$ for $i \in \{1, 2\}$. Then, by definition, $|C^i(b)| - (D[a, b] + |C^i(a)|) \geq 2$, whence $(D[a, b] + |C^i(a)|) \leq |C^i(b)| - 2$. Now, let $t \leq |C^1| + D[a, b]$. By transitivity, $t \leq |C^i(b)| - 2$. However, $C^1(b)[t] = C^2(b)[t]$ for $t \leq |C^1(b)| - 2$ since only the last letter of $C^1(b)$ differs from $C^1(b)$. Therefore, $C^1 \sim_D^0 C^2$ which is a contradiction. ◀

► **Corollary 21.** Let D be a perfect-information matrix, and C a non-initial reachable configuration, there is Player a with $\alpha_a^C = 1$ and for every other Player b , we have $\alpha_b^C = 0$.

We can use this result to establish a characterization of perfect-information matrices. Let C be a reachable non-initial configuration and let Player a be the player that can progress in C . By Corollary 21, we have $\alpha_a^C = 1$ and $\alpha_b^C = 0$ for every Player $b \neq a$. We first make a claim:

▷ Claim 22. $\alpha_{a,b}^C = 1$ and $\alpha_{a,b}^C \leq 0$

31:12 Dependency Matrices for Multiplayer Strategic Dependencies

Using this claim, we obtain $|C(b)| - (D[a, b] + |C(a)|) = 1$ and $|C(a)| - (D[b, a] + |C(b)|) \leq 0$. Then, $D[a, b] + D[b, a] \geq -1$. Since the matrix is progressing, we have $D[a, b] + D[b, a] \leq -1$ and then $D[a, b] + D[b, a] = -1$. In fact, we show that this necessary condition is a precise characterization of the perfect-information matrices.

► **Theorem 23.** *A matrix D is perfect-information if and only if for all players a and b , with $a \neq b$ we have $D[a, b] + D[b, a] = -1$.*

We now show the reciprocal, namely, if D satisfies $D[a, b] + D[b, a] = -1$, then D is perfect-information. The first step is to prove that, the associated meta arena is turn based.

► **Lemma 24.** *For D with $D[a, b] + D[b, a] = -1$ whenever $a \neq b$, there is at most one Player a with $\alpha_a^C \geq 1$ in every configuration C .*

Proof. By contradiction, suppose $\alpha_{a,b}^C \geq 1$ and $\alpha_{b,a}^C \geq 1$. Then, $\alpha_{a,b}^C + \alpha_{b,a}^C \geq 2$. Since $\alpha_{a,b}^C = |C(b)| - (D[a, b] + |C(a)|)$ and $\alpha_{b,a}^C = |C(a)| - (D[b, a] + |C(b)|)$, we obtain $D[a, b] + D[b, a] \leq -2$ which contradicts the assumption on D . ◀

It is left to prove that any two different reachable configurations are not $\stackrel{a}{\sim}_D^0$ -equivalent for any a . We here just give an intuition of the proof by contradiction. Suppose that there are two different reachable configurations C^1 and C^2 such that $C^1 \stackrel{a}{\sim}_D^0 C^2$. We can assume without loss of generality that they are immediate successors of the same reachable configuration C . We compare the progress values of Player a with the one of the only player that can progress in C , and prove that the configurations C^1 and C^2 are equal.

6.2 A Parity Game to solve the Existence of a Winning Strategy

For perfect-information matrices, we establish a reduction from EWS to solving a parity game, thus attaining decidability (Theorem 30). Consider a perfect-information matrix D , a coalition Γ and a formula φ . We define the parity game $G(D, \Gamma, \varphi)$ where the coalition Γ has a winning D -uniform strategy if, and only if, Player 0 has a winning strategy in $G(D, \Gamma, \varphi)$ against Player 1.

The parity game $G(D, \Gamma, \varphi)$ is built up from the deterministic parity automaton \mathcal{A}_φ for φ (see Section 3). Its plays simulate runs of automaton \mathcal{A}_φ on the sequence of growing configurations along a play in the meta arena. Positions in the parity games are pairs composed of states of \mathcal{A}_φ and *buffers*: a *buffer* β is a word vector $(\beta_a)_{a \in \mathcal{P}}$ with at least one empty component. Formally, the set of buffers is:

$$\{(\beta_a)_{a \in \mathcal{P}} \in (\{\top, \perp\}^*)^{\mathcal{P}} \mid \beta_b = \varepsilon \text{ for some } b \in \mathcal{P}\}.$$

The *buffer of a configuration* is the “pending part” of the configuration, namely its greatest suffix that is a buffer.

► **Example 25.** *Consider the perfect-information matrix D_9 below where a reachable configuration C and its buffer are depicted.*

$$D_9 = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{pmatrix} \cdot & 2 & 3 \\ -3 & \cdot & -1 \\ -4 & 0 & \cdot \end{pmatrix} \end{matrix} \quad C = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{matrix} \top & \top & \perp & \top & \perp \\ \top & \perp & \perp & \top & \perp \\ \perp & \perp & \top & \top & \perp \end{matrix} \end{matrix}$$

} buffer

We say that a buffer is *reachable* if it is the buffer of some reachable configuration. We can show that in a reachable configuration, the single player that can progress only depends on the buffer β of this configuration, and we write p_β this player.

We denote by B the set of reachable buffers, by B_\exists the set of buffers β in B where $p_\beta \in \Gamma$, and we let $B \stackrel{\text{def}}{=} B \setminus B_\exists$. Although our matrix is perfect-information, we remark that, by Lemma 20, for the particular case of the empty buffer β^0 (of the initial configuration), Player p_{β^0} might be playing a long move. Moreover, it can be shown that the reachable buffers are finitely many¹ and that their number is exponential in the values of the matrix (this is because the longest component of a reachable buffer is given by the biggest absolute value in the matrix).

We now informally describe the parity game with its two players Player 0 and Player 1. As said, a position in the parity arena is pair (q, β) composed of a state q of the automaton and a buffer β . Position (q, β) belongs to Player 0 whenever $\beta \in B_\exists$, otherwise $\beta \in B_\forall$ and it belongs to Player 1.

In a given position (q, β) , only p_β progresses by choosing a move $u \in \{\top, \perp\}$. We consider the word vector obtained by catenating u to p_β 's component in buffer β , that we write $\beta +_{p_\beta} u$ in the following.

If $\beta +_{p_\beta} u$ is still a buffer we update the position to $(q, \beta +_{p_\beta} u)$. Note that this is always the case for the initial buffer β^0 . Otherwise, the first letter of word vector $\beta +_{p_\beta} u$ is all filled with labels on every component, and thus can be read by automaton \mathcal{A}_φ . We then update the position to the new current state of \mathcal{A}_φ and to the buffer obtained by removing the first letter of $\beta +_{p_\beta} u$ (which is a buffer as β is not initial).

Formally, the parity game is the following.

► **Definition 26.** *Given a perfect-information matrix D , a coalition of players Γ and a deterministic parity automaton $\mathcal{A}_\varphi = (Q, q_0, \Sigma, \delta, \text{par})$ with $\Sigma = \{\top, \perp\}^P$, we define the parity game $G(D, \Gamma, \varphi) = \langle P_0, P_1, s_0, \rightarrow, \text{par}_G \rangle$ where:*

- $P_0 = Q \times B_\exists$ is the set of positions for Player 0,
- $P_1 = Q \times B_\forall$ is the set of positions for Player 1,
- $s_0 = (q_0, \beta^0)$ is the initial position,
- $(q, \beta) \rightarrow (q', \beta')$ when there is a legitimate move u for p_β in the meta arena such that:
 1. either $\beta +_{p_\beta} u$ is a buffer and $q = q'$ and $\beta' = \beta +_{p_\beta} u$.
 2. or $q' = \delta(q, (\beta +_{p_\beta} u)[0])$ and $\beta' = (\beta +_{p_\beta} u)[1:]$ and p_β is the only player s.t. $\beta_{p_\beta} = \varepsilon$;
- $\text{par}_G(q, \beta) = \text{par}(q)$, that is the priority of a position (q, β) is the priority of the state q in the automaton \mathcal{A}_φ .

Note that the number of positions in the parity game is the product of the number of states in the automaton and the number of buffers, and that the game has the same priorities as the automaton.

► **Proposition 27** (For a perfect-information matrix D). *$\langle D, \Gamma, \varphi \rangle$ is a positive instance of EWS if, and only if, Player 0 has a winning strategy in $G(D, \Gamma, \varphi)$.*

Proposition 27 gives us an upper bound complexity of EWS by the following algorithm.

1. Compute the deterministic parity automaton \mathcal{A}_φ (accepting the models of φ);
2. Compute the parity game $G(D, \Gamma, \varphi)$;
3. Solve $G(D, \Gamma, \varphi)$.

¹ Actually the set of reachable configurations is a regular language that can be recognized by a word automaton with buffers as states.

31:14 Dependency Matrices for Multiplayer Strategic Dependencies

In the following, the *size* of the matrix D is the quantity $|D| = \sum_{a \neq b} |D[a, b]|$. Observe that we can build \mathcal{A}_φ by using the Vardi-Wolper construction [17] with the Safra-like translation from Büchi to parity acceptance condition [11], so that parity game of Step 2 has $O(2^{2^{|\varphi|}} \times 2^{2^{|D|}})$ positions and $O(2^{|\varphi|})$ priorities, hence a 2-EXPTIME decision procedure for EWS.

The next subsection, we show that this algorithm is essentially optimal by a reduction of the Church Synthesis problem.

6.3 Reduction from the Church Synthesis problem

For the lower bound, we reduce the Church Synthesis for LTL properties [5, 12]. Our Example 5 (page 4) illustrates the reduction.

► **Definition 28.** *Given a coalition Γ , a Church matrix is a matrix D where for any two players $a \neq b$, we have:*

$$D[a, b] = \begin{cases} 0 & \text{if } a \notin \Gamma \text{ and } b \in \Gamma \\ -1 & \text{otherwise} \end{cases}$$

In essence, for Church matrices, players have the same knowledge about the current configuration, allowing them to foresee their allies moves.

Observe that Church matrices may not be perfect-information, since the moves of every player in a team (coalition or opponents) are concurrent. Nonetheless, we can “transform” any Church matrix D into a linear sized perfect-information Round Robin matrix D' (see Example 3 and Definition 29) such that $\langle D, \Gamma, \varphi \rangle$ is a positive instance of EWS if, and only if, $\langle D', \Gamma, \varphi \rangle$ is a positive instance of EWS.

► **Definition 29.** *A Round Robin matrix is a matrix D such that there exists a total order \prec over \mathcal{P} , where*

$$D[a, b] = \begin{cases} -1 & \text{if } a \prec b \\ 0 & \text{otherwise} \end{cases}$$

The total order \prec describes the order in which players will play (the player that is minimal for \prec plays first). Remark that $D[a, b] + D[b, a] = -1$ for any two players $a \neq b$ then every Round Robin matrix is perfect-information (by Theorem 23). Given a Church matrix, we can choose order \prec so that all players in the coalition play before their opponents.

By summing up, we polynomially reduce² a Church synthesis problem to a EWS problem for a Church matrix and that is in turn linearly reduced to a EWS problem for a Round Robin matrix. From this latter reductions, we can state the following.

► **Theorem 30.** *EWS for perfect-information matrices is 2-EXPTIME-complete in the size of the LTL formula.*

In the next section, we extend the class of perfect-information matrices to allow some matrices with infinite values, while keeping the decidability of EWS for the resulting superclass. In particular, QPTL matrices (see Example 4 on page 4) falls into this class.

² The size of the Church matrix is quadratic in the number of propositions of the Church synthesis problem

6.4 Perfect-Information Matrices with Possibly Infinite Values

The proof of Proposition 27 can be extended to QPTL formulas instead of LTL formulas. The following example illustrates a procedure for matrices with infinite values that yields a generalization of the perfect-information property (see Definition 32).

► **Example 31.** Consider the following matrices where the latter is perfect-information:

$$D_8 = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{pmatrix} \cdot & +\infty & +\infty \\ -\infty & \cdot & -1 \\ -\infty & 0 & \cdot \end{pmatrix} \end{matrix} \quad \text{and} \quad D'_8 = \begin{matrix} & \begin{matrix} b & c \end{matrix} \\ \begin{matrix} b \\ c \end{matrix} & \begin{pmatrix} \cdot & -1 \\ 0 & \cdot \end{pmatrix} \end{matrix}$$

According to D_8 , Player a depends on the whole labeling of Player b and Player c . Given an LTL formula φ and say coalition $\{a, b\}$, we can answer the EWS on instance $\langle D_8, \{b\}, \varphi \rangle$ as follows: we can first answer EWS for $\langle D'_8, \{b\}, \exists a \varphi \rangle$ (since D'_8 is perfect-information). If no, then return no for $\langle D_8, \{b\}, \varphi \rangle$. Otherwise, each outcome of the winning strategy for Player 0 in $\langle D'_8, \{b\}, \exists a \varphi \rangle$ reflects a play ρ in $\langle D'_8, \{b\}, \exists a \varphi \rangle$. From play ρ , exhibit a unique accepting run in $\mathcal{A}_{\exists a \varphi}$. By tracing back this run inside \mathcal{A}_φ , reconstruct Player a 's response to the play ρ .

The procedure employed in Example 31 applies to arbitrary matrices as long as they fulfill the Definition 32.

► **Definition 32.** An arbitrary matrix D is perfect-information if for any $a \neq b$:

1. $D[a, b] \in \mathbb{Z}$ implies $D[a, b] + D[b, a] = -1$;
2. $D[a, b] \in \{-\infty, +\infty\}$ implies $D[a, b] = -D[b, a]$;
3. $D[a, b] = +\infty$ implies $D[a, c] \in \{-\infty, +\infty\}$, for all $c \neq a$.

Observe that the procedure is in fact non-elementary in the number of players with $+\infty$ dependencies. Moreover, since the validity problem for QPTL [14] reduces to EWS for arbitrary perfect-information matrices, we have the following.

► **Theorem 33.** EWS is non-elementary for arbitrary perfect-information matrices.

7 Conclusion

We presented the expressive framework of dependency matrices that can capture several game settings such as concurrent and turn-based games [2], (two-player) delay games [8, 9, 16], logic QPTL [14], and Church Synthesis Problem [5].

We proved that the existence of a winning strategy for a coalition to achieve an LTL formula (EWS) is undecidable for arbitrary matrices.

We then exhibited the subclass of perfect-information bounded-value matrices for which the problem EWS is 2-EXPTIME-complete in the size of the formula.

Finally, we extended the class of perfect-information matrices with a narrow use of infinite dependencies allowing to re-use known techniques of automata projection for QPTL. For these matrices, EWS becomes non-elementary. Still our complexity analysis of EWS needs being refined regarding the matrix parameter: we do not know yet the lower bound complexity when the LTL formula is fixed.

A first track to continue this work concerns EWS for the whole class of bounded-value matrices. We conjecture it is decidable, since, for a bounded-value matrix, each k -indistinguishable equivalence class of a reachable configurations has a bounded size.

A second track regards our transformation of Church matrices into perfect-information Round Robin ones. We believe that our approach can generalize to a class of bounded-values matrices enlarging the one of Church matrices.

References

- 1 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986. URL: <https://www.worldcat.org/oclc/12285707>.
- 2 Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.
- 3 Robert Berger. *The undecidability of the domino problem*. American Mathematical Soc., 1966.
- 4 Krishnendu Chatterjee, Thomas A Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010.
- 5 Bernd Finkbeiner. Synthesis of reactive systems. *Dependable Software Systems Engineering*, 45:72–98, 2016.
- 6 David Gale and Frank M Stewart. Infinite games with perfect information. *Contributions to the Theory of Games*, 2(245-266):2–16, 1953.
- 7 Edward F. Grove. Online bin packing with lookahead. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 430–436. ACM/SIAM, 1995. URL: <http://dl.acm.org/citation.cfm?id=313651.313781>.
- 8 Felix Klein and Martin Zimmermann. What are strategies in delay games? borel determinacy for games with lookahead. *arXiv preprint*, 2015. [arXiv:1504.02627](https://arxiv.org/abs/1504.02627).
- 9 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12, 2017.
- 10 Gary Peterson, John Reif, and Salman Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7-8):957–992, 2001.
- 11 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3, 2007.
- 12 A Pnueli and R Rosner. On the synthesis of reactive systems. *POPL, Austin, Texas*, pages 179–190, 1989.
- 13 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.
- 14 A Prasad Sistla, Moshe Y Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.
- 15 Aravinda Prasad Sistla. *Theoretical issues in the design and verification of distributed systems*. Harvard University, 1983.
- 16 Wolfgang Thomas, Lukasz Kaiser, and Michael Holtmann. Degrees of lookahead in regular infinite games. *Logical Methods in Computer Science*, 8, 2012.
- 17 Moshe Y Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32(2):183–221, 1986.

A Proofs of Section 4

Given a matrix D , we say that a player a is *eventually blocked* if there is a natural $k \in \mathbb{N}$ such that for all reachable configurations C , we have $|C(a)| \leq k$. First we prove a lemma on non-negative cycles. Intuitively, this lemma helps to find a player that can never progress in a non-negative cycle.

Given two vertices c_i and c_j of a non-negative cycle $c = (c_0, \dots, c_{|c|})$ (with $c_0 = c_{|c|}$), we denote by w_i the label $r(c_i, c_{i+1})$ and $W_{i,j}$ the circular sum of the labels between c_i and c_j :

- If $i < j$, then $W_{i,j} = w_i + \dots + w_{j-1}$;
- else, $W_{i,j} = W_{i,|c|} + W_{0,j}$.

Remark that for every index i , $W_{i,i}$ is the sum of all labels of the cycle. So, for a non-negative cycle, $W_{i,i} \geq 0$. Furthermore, we denote by W_i^* the minimal sum $W_i^* = \min_j(W_{i,j})$.

► **Lemma 34.** *Given a matrix D with a non-negative cycle $c = (c_0, \dots, c_{|c|})$, for all player c_i in the cycle, for all reachable configuration C , we have the following.*

$$|C(c_i)| \leq \max(0, -W_i^*)$$

Proof. We do the proof by induction on reachable configuration.

(base case) In the initial configuration C^0 , the property is obvious.

(inductive case) Consider a configuration $C' = \Delta(C, (u_a)_{a \in \mathcal{P}})$ for some joint move $(u_a)_{a \in \mathcal{P}}$ and a reachable configuration C such that $|C(c_i)| \leq \max(0, W_i^*)$ for every $c_i \in c$. Consider $i \in \llbracket 0, \dots, |c| \rrbracket$.

$$\begin{aligned} |C'(c_i)| &= |C(c_i)| + \alpha_{c_i}^C \\ &= |C(c_i)| + \max(0, \min_{a \neq c_i}(\alpha_{c_i, a}^C)) \\ &\leq |C(c_i)| + \max(0, \alpha_{c_i, c_{i+1}}^C) \\ &\leq |C(c_i)| + \max(0, |C(c_{i+1})| - D[c_i, c_{i+1}] - |C(c_i)|) \\ &\leq \max(|C(c_i)|, |C(c_{i+1})| - w_i) \\ &\leq \max(\max(0, -W_i^*), \max(-w_i, -W_{i+1}^* - w_i)) \end{aligned}$$

The last inequality is obtained thanks to the inductive hypothesis. We now prove that $-W_i^* \geq \max(-w_i, -W_{i+1}^* - w_i)$. Since $w_i = W_{i, i+1}$, we have $-W_i^* \geq -w_i$. Let $j_0 \in \llbracket 0, \dots, |c| \rrbracket$ such that $W_{i+1}^* = W_{i+1, j_0}$. We now do a case study on j_0 .

- if $j_0 = i + 1$, then, because c is non-negative, $-W_{i+1}^* - w_i \leq -w_i$.
- if $j_0 \neq i + 1$, then, $-W_{i+1}^* - w_i = -W_{i, j_0} \leq -W_i^*$

Then we have $|C'(c_i)| \leq \max(0, -W_i^*)$.

We have proven the property by induction. ◀

We now state a lemma to prove the other way around: that non-negative cycles are necessary for a matrix not to be progressing.

► **Lemma 35.** *Given a bounded matrix D , if there is a player that is eventually blocked, then every player is eventually blocked.*

Proof. Consider a bounded matrix D where there is a player a and a natural number $k \in \mathbb{Z}$ such that for every reachable configuration C , we have $|C(a)| \leq k$. We prove by induction on reachable configurations that for each Player b , we have $|C(a)| \leq \max(0, k - D[b, a])$.

(base case) In the initial configuration C^0 , the property is obvious.

(inductive case) Consider a configuration $C' = \Delta(C, (u_a)_{a \in \mathcal{P}})$ for some joint move $(u_a)_{a \in \mathcal{P}}$ and a configuration C such that $|C(a)| \leq k - D[b, a]$ for every b . Then, for every Player b we have the following.

$$\begin{aligned}
 |C'(b)| &= |C(b)| + \alpha_b^C \\
 &= |C(b)| + \max(0, \min_{c \neq b}(\alpha_{b,c}^C)) \\
 &\leq |C(b)| + \max(0, \alpha_{b,a}^C) \\
 &\leq |C(b)| + \max(0, |C(a)| - D[b, a] - |C(b)|) \\
 &\leq \max(|C(b)|, |C(a)| - D[b, a]) \\
 &\leq \max(\max(0, k - D[b, a]), k - D[b, a]) \\
 &\leq \max(0, k - D[b, a])
 \end{aligned}$$

We have proven the property by induction. \blacktriangleleft

Now, when all players are blocked, we use the next well known result of graph theory to find our non-negative cycle.

► **Lemma 36.** *Given a directed graph G with no self loop, if every vertex is the source of an edge, then, there is a cycle in the graph.*

Now, we can give a proof for Proposition 12.

► **Proposition 12.** *A matrix D is progressing if, and only if, its dependency graph G_D has no non-negative-weighted cycle.*

Proof. Given a matrix D with a non-negative cycle, then, by Lemma 34, we immediately have that every player in the cycle is eventually blocked.

In a second time, consider a matrix D that is not progressing. Then, by Lemma 35, for every player a , there is an integer k_a such that for every reachable configuration C , we have $|C(a)| \leq k_a$. Consider a configuration C such that every player is blocked. Then, by immediate contradiction, for every player a , there is a player $q_a \neq a$ such that $\alpha_{a,q_a}^C \leq 0$ (otherwise, there would be a player that can progress). The graph $G = \langle V, E \rangle$ with the vertices $V = \mathcal{P}$ are the players of the matrix and the edges are defined as $E = \{(a, q_a) \mid a \in \mathcal{P}\}$. Since, G is a directed graph with no self loop, by Lemma 36, there is a cycle $c = (c_0, \dots, c_{|c|})$ in the graph thus, for every $i \in \llbracket 0, \dots, |c| - 1 \rrbracket$, we have $c_{i+1} = q_{c_i}$ and $c_{|c|} = c_0$. We have the following.

$$\begin{aligned}
 \sum_{i=0}^{|c|-1} \alpha_{c_i, c_{i+1}}^C &= \sum_{i=0}^{|c|-1} |C(c_{i+1})| - D[c_i; c_{i+1}] - |C(c_i)| \\
 &= - \sum_{i=0}^{|c|-1} D[c_i; c_{i+1}]
 \end{aligned}$$

Since $\alpha_{c_i, c_{i+1}}^C \leq 0$ for every $i \in \llbracket 0, \dots, |c| \rrbracket$, we have proven that c is a non-negative cycle. \blacktriangleleft

B Proofs of Section 6

We now address the proof of Theorem 23. The first direction states that a perfect-information matrix D satisfies that for every different Players a and b , we have $D[a, b] + D[b, a] = -1$ and is presented in Section 6.1. We just need to prove Claim 22. Recall that C is a reachable non-initial configuration and Player a is the player that can progress in C . By Corollary 21, we have $\alpha_a^C = 1$ and $\alpha_b^C = 0$ for every player $b \neq a$. We consider a player $b \neq a$.

▷ **Claim 22.** $\alpha_{a,b}^C = 1$ and $\alpha_{a,b}^C \leq 0$

Proof. First we prove that $\alpha_{a,b}^C \leq 0$. Remark that, in C , Player a has two legitimate moves: \top and \perp . Let $C^1 = \Delta(C, (\top)_a)$ and $C^2 = \Delta(C, (\perp)_a)$. Note that C^1 and C^2 are reachable. Toward contradiction, assume that $\alpha_{a,b}^C > 0$. We prove the contradiction $C^1 \stackrel{b}{\sim}_D C^2$. We

have $\alpha_{a,b}^C = |C(a)| - (D[b,a] + |C(b)|) > 0$. Then $|C(a)| > D[b,a] + |C(b)|$. Therefore, for all $t \leq D[b,a] + |C(b)|$, we have $t < |C(a)|$. And because $C^1(a)(t) = C(a)(t) = C^2(a)(t)$, we conclude that $C^1 \stackrel{b}{\sim}_D C^2$.

Then we prove that $\alpha_{a,b}^C = 1$. By definition, $\alpha_{a,b}^C \geq 1$. Toward contradiction, suppose $\alpha_{a,b}^C > 1$. Let C' be the configuration defined by $C'(c) = C(c)$ for all $c \neq b$ and $C'(b) = \text{fliplast}(C(b))$. We have that C' is reachable and, by the same kind of reasoning than previous point, we have $C \stackrel{a}{\sim}_D C'$, which is in contradiction with the assumption on D . \triangleleft

Let us prove the other direction, namely that a matrix D is perfect-information if $D[a,b] + D[b,a] = -1$ for every pair of different Players a and b . Lemma 24 states that the meta game of such a matrix is turn based. Now, we need to prove that two different reachable configurations are not 0-indistinguishable. To do so, we first show two results on 0-indistinguishable relations. These results allow us to consider the “first time” at which two configurations diverge while being 0-indistinguishable.

► **Lemma 37.** *Given a matrix D and two reachable configurations C and C' , with $(C^k)_{k \leq n}$ and $(C'^k)_{k \leq m}$ such that $C^n = C$ and $C'^m = C'$. If $n \geq m$ then, for every Player a we have $|C(a)| \geq |C'(a)|$.*

Proof. The proof is done by induction on $n - m$.

(base case) If $n = m$, we do an induction on n .

(base case) If $n = m = 0$, then $C = C' = C^0$, the property is immediate.

(induction) Suppose the property holds for some $n, m \in \mathbb{N}$ with $n = m$. Consider C and C' reachable. There are $(C^k)_{k \leq n+1}$ and $(C'^k)_{k \leq m+1}$ with $C^{n+1} = C$ and $C'^{m+1} = C'$. By inductive hypothesis, we have $|C^n(a)| = |C'^m(a)|$ for every Player a . Then, $\alpha_a^{C^n} = \alpha_a^{C'^m}$ and so, $|C(a)| = |C'(a)|$.

(induction) Suppose the property holds for some $n, m \in \mathbb{N}$ with $n \geq m$. Consider C and C' reachable: there are $(C^k)_{k \leq n+1}$ and $(C'^k)_{k \leq m}$ with $C^{n+1} = C$ and $C'^m = C'$. By inductive hypothesis, for every Player a $|C^n(a)| \geq |C'^m(a)|$ and because $|C^{n+1}(a)| \geq |C^n(a)|$, we have $|C^{n+1}(a)| \geq |C'^m(a)|$.

We have proved the property by induction. \triangleleft

► **Lemma 38.** *Given a dependency matrix D , and two reachable configurations C and C' that are non-initial, if $C \stackrel{a}{\sim}_D C'$, then, one of the following holds.*

1. $\Delta^{-1}(C) \stackrel{a}{\sim}_D C'$
2. $C \stackrel{a}{\sim}_D \Delta^{-1}(C')$
3. $\Delta^{-1}(C) \stackrel{a}{\sim}_D \Delta^{-1}(C')$

Proof. Consider two reachable configurations C and C' such that $C \stackrel{a}{\sim}_D C'$. By definition, $|C(a)| = |C'(a)|$, and for every Player $b \neq a$, every $t \leq |C(a)| + D[a,b]$, we have $C(b)(t) = C'(b)(t)$. Since C and C' are reachable, there are two sequences of successive configurations $(C^k)_{k \leq n}$ and $(C'^k)_{k \leq m}$ such that $C^n = C$ and $C'^m = C'$. By symmetry we can assume $n \geq m$. We do a case study.

If $n = m$, then $\Delta^{-1}(C)$ and $\Delta^{-1}(C')$ are also reachable and their sequences have the same length. By Lemma 37 for every player b , we have that $|\Delta^{-1}(C)(b)| = |\Delta^{-1}(C')(b)|$. Immediately, $|\Delta^{-1}(C)(a)| = |\Delta^{-1}(C')(a)|$ and because $C \stackrel{a}{\sim}_D C'$, for every $t \leq |\Delta^{-1}(C)(a)| + D[a,b] \leq |C(a)| + D[a,b]$, we have $\Delta^{-1}(C)(b)(t) = C(b)(t) = C'(b)(t) = \Delta^{-1}(C')(b)(t)$. Hence, $\Delta^{-1}(C) \stackrel{a}{\sim}_D \Delta^{-1}(C')$

31:20 Dependency Matrices for Multiplayer Strategic Dependencies

If $n > m$ then $\Delta^{-1}(C)$ is reachable with a sequence of length $n - 1 \geq m$ as $\Delta^{-1}(C) = C^{m-1}$ then, by Lemma 37, for every player b , we have $|\Delta^{-1}(C)(b)| \geq |C'^m(b)|$. And, because $C \stackrel{a_0}{\sim}_D C'$, we have $|C(a)| = |C'(a)|$, then $|\Delta^{-1}(C)(a)| = |C'(a)|$. Furthermore, for every $t \leq |\Delta^{-1}(C)(a)| + D[a, b]$, if both $\Delta^{-1}(C)(b)(t)$ and $C'(b)(t)$ are defined, we have $\Delta^{-1}(C)(b)(t) = C'(b)(t) = C'^m(b)(t)$. Finally, we prove that $C(b)(t)$ is defined if and only if $C'(b)(t)$ is defined. By Lemma 37, we already have that $|\Delta^{-1}(C)(b)| \geq |C'^m(b)|$, and if $\Delta^{-1}(C)(b)(t)$ is defined, so is $C(b)(t)$, and by hypothesis, so is $C'^m(b)(t)$.

We have proved the property. \blacktriangleleft

We now prove the theorem.

► **Theorem 23.** *A matrix D is perfect-information if and only if for all players a and b , with $a \neq b$ we have $D[a, b] + D[b, a] = -1$.*

Proof. The first direction is presented in Section 6.1. For the other direction, consider D such that $D[a, b] + D[b, a] = -1$. By Lemma 24, there is only one player that can progress in any reachable configuration. Is left to prove that for any two reachable configurations C and C' with $C \neq C'$, for every player a , we have $C \not\stackrel{a_0}{\sim}_D C'$. The proof is done by contradiction.

Suppose that there are two different reachable configurations C^1 and C^2 with $C^1 \stackrel{a_0}{\sim}_D C^2$. Then, by Lemma 38, we can assume that $\Delta^{-1}(C^1) = \Delta^{-1}(C^2) = C$. Let b the player that can progress in C . Then, for all $c \neq b$, we have $C(c) = C^1(c) = C^2(c)$ and, for all $t < |C(b)|$, we have $C(b)(t) = C^1(b)(t) = C^2(b)(t)$. Because $C^1 \neq C^2$, then we necessarily have the following.

$$C^1(b)(t_0) \neq C^2(b)(t_0) \text{ for some } t_0 \in \{|C(b)| - \alpha_b^C, |C(b)| - 1\} \quad (2)$$

Since $\alpha_{b,a}^C \geq \alpha_b^C$, we have $|C(a)| - (D[b, a] + |C(b)|) \geq \alpha_b^C$. As Player a does not progress in C , we obtain $|C^1(a)| - (D[b, a] + |C(b)|) \geq \alpha_b^C$. By hypothesis $D[b, a] = -1 - D[a, b]$, we have $|C^1(a)| + 1 + D[a, b] - |C(b)| \geq \alpha_b^C$ and because $|\Delta^{-1}(C^1)(b)| + \alpha_b^C = |C^1(b)|$, we have:

$$|C^1(a)| + D[a, b] \geq |C^1(b)| - 1 \quad (3)$$

Finally, since $C^1 \stackrel{a_0}{\sim}_D C^2$, we have that $C^1(b)(t) = C^2(b)(t)$ for every $t \leq |C^1(a)| + D[a, b]$. In particular, thanks to Equation (3), we can take $t = t_0$, and we have $C^1(b)(t_0) = C^2(b)(t_0)$, in contradiction with Equation (2). \blacktriangleleft

C Reduction from Church matrices to Round Robin matrices

We now prove the claim made in Section 6.3 that Church matrices can be reduced to Round Robin matrices.

From a Church matrix D for a coalition Γ , we define a Round Robin matrix $\text{Rob}(D)$ as follows. We take an arbitrary order \prec on the players such that for every Player $a \in \Gamma$ and every Player $b \notin \Gamma$, it holds $a \prec b$. $\text{Rob}(D)$ is the Round Robin matrix for this order.

► **Lemma 39.** *Given an LTL formula φ and a Church matrix D , there is a winning joint D -uniform strategy iff there is a winning joint strategy $\text{Rob}(D)$.*

Proof. Suppose that there is a winning joint strategy F for Γ that is $\text{Rob}(D)$ -uniform. Only the moves of players of the coalition can make F non- D -uniform. But, given the joint strategy for the whole coalition, we cannot reach two configurations that are equal on everything except a labeling of a player of the coalition because our strategies are deterministic. Thus, in practice, F is D -uniform.

Conversely, consider two configurations C^1 and C^2 such that $C^1 \stackrel{a}{\sim}_{\text{Rob}(D)} C^2$ for some $a \in \Gamma$. We denote by k the length of $C^1(a)$. Then for every $b \in \mathcal{P}$, if $a \prec b$, we have $C^1(b)[:k-1] = C^2(b)[:k-1]$ and otherwise, $C^1(b)[:k] = C^2(b)[:k]$. Because we chose the order so that $a \prec b$ for every $a \in \Gamma$ and $b \notin \Gamma$, we have that $C^1 \stackrel{a}{\sim}_D C^2$. Thus every strategy D -uniform is $\text{Rob}(D)$ -uniform. \blacktriangleleft

D Proof of Theorem 33

► **Theorem 33.** *EWS is non-elementary for arbitrary perfect-information matrices.*

Proof. In this proof, we use the notation $F_{\upharpoonright \mathcal{P} \setminus \{a\}}$ to denote the function F restrained to the domain $\mathcal{P} \setminus \{a\}$. Given a dependency matrix D , we decompose the set of Players \mathcal{P} as follows.

$$\begin{aligned} \mathcal{P}_\infty &:= \{a \in \mathcal{P} \mid \text{There is } b \in \mathcal{P} \text{ such that } D[a, b] = +\infty\} \\ \mathcal{P}_Z &:= \{a \in \mathcal{P} \mid \text{For all } b \in \mathcal{P} \text{ it holds } D[a, b] < +\infty\} \end{aligned}$$

We reason by induction on the size of \mathcal{P}_∞ . The base case is $|\mathcal{P}_\infty| = 0$. In this case, we can apply Theorem 30.

Suppose that we can decide the EWS problem for matrices with $|\mathcal{P}_\infty| = n$ for some $n \in \mathbb{N}$. We now prove that we can decide the problem for matrices with $|\mathcal{P}_\infty| = n + 1$. Consider a matrix D such that $|\mathcal{P}_\infty| = n + 1$, a coalition Γ and a QPTL formula ψ . We define an order \prec on \mathcal{P}_∞ that is given as follows. $a \prec b$ iff for all $c \in \mathcal{P}$, if $D[b, c] = +\infty$, then $D[a, c] = +\infty$. Intuitively, $a \prec b$ means that Player a is to play after Player b . Consider Player a , the smallest player for this order. For all players b different than a we have $D[a, b] = +\infty$. We now construct a new instance of the problem by projecting out Player a .

If $(a \in \Gamma)$ we state $\Gamma' = \Gamma \setminus \{a\}$ and $\psi' = \exists a. \psi$. By inductive hypothesis, we can decide whether there is a joint strategy winning for the entry $\langle D', \Gamma', \psi' \rangle$. Let us prove that $\langle D', \Gamma', \psi' \rangle$ is a positive instance iff $\langle D, \Gamma, \psi \rangle$ is a positive instance. If there is a joint strategy F' for the coalition winning for the entry $\langle D', \Gamma', \psi' \rangle$, then, for every play (C_1^n, \dots, C_k^n) in the meta game of D' , the assignment λ which is the limit of that play satisfy $\langle D', \Gamma', \psi' \rangle$. Therefore, there is an infinite word u_λ such that $\lambda[a \mapsto u_\lambda]$ satisfies ψ' . Then we define the joint strategy F as follows. For every Player $b \neq a$, we set $F_b = F'_b$. For Player a , consider a configuration C such that $\alpha_a^C > 0$. Because of the dependencies of Player a , it holds that $|C(b)| = +\infty$ for every Player $b \neq a$. Let λ be the temporal assignment on $\mathcal{P} \setminus \{a\}$ defined by C . We set $F_a(C) = u_\lambda$. This joint strategy is winning. The converse follows the same idea: if there is F winning for the entry (D, Γ, ψ) then, the joint strategy $F' = F_{\upharpoonright \mathcal{P} \setminus \{a\}}$ is winning for the entry (D', Γ', ψ') .

If $(a \notin \Gamma)$ we state $\psi' = \forall a. \psi$. We then decide the instance $\langle D', \Gamma, \psi' \rangle$. Joint strategies for the coalition translate naturally between the two instances. If F' is winning for $\langle D', \Gamma, \psi' \rangle$, then every play in the outcome of F' satisfies $\forall a \psi$. Then, by defining $F(C) = F'(C_{\upharpoonright \mathcal{P} \setminus \{a\}})$ we define a winning strategy for $\langle D, \Gamma, \psi \rangle$. The converse is similar. \blacktriangleleft

Semilinear Representations for Series-Parallel Atomic Congestion Games

Nathalie Bertrand  

Univ Rennes, Inria, CNRS, IRISA, France

Nicolas Markey  

Univ Rennes, Inria, CNRS, IRISA, France

Suman Sadhukhan 

Univ Rennes, Inria, CNRS, IRISA, France

University of Haifa, Israel

Ocan Sankur  

Univ Rennes, Inria, CNRS, IRISA, France

Abstract

We consider atomic congestion games on series-parallel networks, and study the structure of the sets of Nash equilibria and social local optima on a given network when the number of players varies. We establish that these sets are definable in Presburger arithmetic and that they admit semilinear representations whose all period vectors have a common direction. As an application, we prove that the prices of anarchy and stability converge to 1 as the number of players goes to infinity, and show how to exploit these semilinear representations to compute these ratios precisely for a given network and number of players.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Solution concepts in game theory

Keywords and phrases congestion games, Nash equilibria, Presburger arithmetic, semilinear sets, price of anarchy

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.32

Funding This research was partially supported by ISF grant no. 1679/21.

1 Introduction

Network congestion games are used to model situations in which agents share resources such as routes or bandwidth [31], and have applications in communication networks (e.g. [1]). We consider the *atomic* variant of these games, where each player controls one unit of flow and must assign it to a path in the network. All players using an edge then incur a cost (a.k.a. latency) that is a nondecreasing function of the number of players using the same edge. Since all players try to minimize their own cost, this yields a noncooperative multiplayer game. It is well known that Nash equilibria exist in these games [31] but that they can be inefficient, that is, a global measure such as the total cost, or the makespan may not be minimized by Nash equilibria [30].

To quantify this inefficiency, [27] introduced the notion of *price of anarchy (PoA)*, which is the ratio of the cost of the worst Nash equilibrium and the social optimum. Here, social optimum refers to the sum of the individual costs. A tight bound of $\frac{5}{2}$ on this ratio was given in [3, 9]. Various works have studied bounds on the PoA for restricted classes of graphs or types of cost functions; see [28]. While the price of anarchy is interesting to understand behaviors that emerge in a system from a worst-case perspective, the best-case is also interesting if, for instance, the network designer is able to select a Nash equilibrium.



© Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 32; pp. 32:1–32:20



Leibniz International Proceedings in Informatics

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

The *price of stability* (*PoS*) is thus the ratio between the cost of the best Nash equilibrium and the social cost, and was studied in [2]; a bound of $1 + \sqrt{3}/3$ was given in the atomic case; see [7, 8, 2].

These bounds have been studied for restrictions of this problem such as particular classes of graphs. One such class is *series-parallel networks* which are built using single edges, or parallel and serial composition of smaller series-parallel networks (see e.g. [34]). On these networks with affine cost functions, [24] reports that PoA is between $\frac{27}{19}$ and 2, where a previous known lower bound due to [19] was $\frac{15}{11}$. [19] also proves an upper bound on PoA for *extension-parallel* networks which are a strict subclass of series-parallel networks.

While tight bounds are known for atomic network congestion games and even for particular subclasses, this does not help one to evaluate the price of anarchy of a given specific game with a given number of players. In fact, the upper bounds mentioned above are obtained by building particular networks, and these are shown to be tight by exhibiting families of instances in which both the networks and the number of players vary. One of our objectives is to provide tools to analyze a given network congestion game, by computing both ratios precisely for varying numbers of players. We are interested both in the case of a given number of players, and in the case of the limit behavior. Note that we are considering a hard problem since computing *extreme* Nash equilibria, that is, best and worst ones, is NP-hard in networks with only three and two players respectively [33].

In this paper, we consider series-parallel networks with linear cost functions and establish interesting properties of their Nash equilibria and social optima. We start with the observation that Nash equilibria and *locally* social optima can be expressed in Presburger arithmetic; it follows that these sets admit *semilinear* representations [22]. Our main result is that that these semilinear sets have a particular structure. In fact, the flows (i.e. edge loads) induced by Nash equilibria and local social optima admit semilinear representations with a common direction for all period vectors, which is moreover efficiently computable. We call this direction the *characteristic vector* of the network. Intuitively, this vector determines how the flow evolves in Nash equilibria and social local optima as the number of players increases.

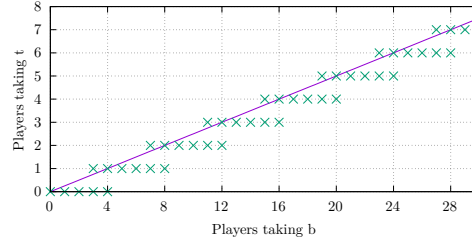
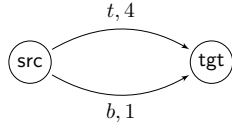
We believe that the form of these representations and the characteristic vector have an interest on their own. We give one application of these representations here, namely, that the PoA and the PoS both tend to 1 as the number of players goes to infinity in series-parallel networks with linear cost functions with positive coefficients. This result was proven recently [37]; we provide here new techniques and thus an alternative proof. Observe that a similar result holds in nonatomic congestion games [11, 10] (see Section 6 for a discussion).

We also illustrate how these semilinear representations allow one to study the evolution of PoA and PoS for a given network. The computation of these representations is an expensive step, but once this is done, thanks to the particular form of the representations, for any n , one can easily query the exact value for PoA and PoS in the network instantiated with n players. One can thus analyze both ratios precisely and specifically for a given network as a function of n , while the limits will always be 1.

Illustrating Example

Let us illustrate our results on a simple example. Consider the network with two parallel links in Figure 1a. There are n players who would like to go from **src** to **tgt**, by taking either the bottom edge (b) with the cost function $x \mapsto x$, or the top edge (t) with cost function $x \mapsto 4x$. The cost function determines the cost each player pays when taking a given edge, and it is a function of the total number of players using the same edge. For instance, for $n = 4$, assume

that 3 players use b , and 1 uses t . Then, each player using b pays a cost of 3, while the only player using t pays 4. Here, a strategy profile can be seen as a pair $(k, n - k)$ determining how many players take t and how many take b .



(a) A network with two parallel links where each player can choose either the top edge (t) with cost $x \mapsto 4x$, or the bottom edge (b) with cost $x \mapsto x$. (b) Nash equilibria in the network on the left for varying total numbers of players. There is a point at coordinates (x, y) if a strategy profile assigning x players to b , and y players to t is a Nash equilibrium.

Figure 1 Analysis of a simple network congestion game.

Intuitively, in Nash equilibria, the number of players taking b should be *roughly* four times the number of them taking t ; so $\frac{4}{5}$ of them should take b , and $\frac{1}{5}$ of them should take t . This would make sure that both edges have identical cost, and make profitable deviations impossible. Although this intuition holds in the nonatomic case, players cannot always be split with this proportion in the atomic case, as in the case of $n = 4$ above; and there are indeed equilibria that do not match this proportion exactly. Figure 1b shows the Nash equilibria in this game, while the line with direction $(4, 1)$ shows the ideal distribution (as in the nonatomic case). Not all Nash equilibria are on this line, but one can notice that they do form a tube around this line that go in the same direction. Formally, our results determine that the Nash equilibria form the semilinear set $B_{NE} + \vec{\delta} \cdot \mathbb{N}$, where $\vec{\delta} = (4, 1)$, and $B_{NE} = \{(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (3, 1)\}$. In other terms, it is the union of the integer points of six lines with the same direction vector $\vec{\delta}$.

Similarly, we describe the set of *locally* social optimal profiles and show that it admits a semilinear representation. Locally optimal profiles are those in which the social cost cannot be decreased by changing the strategy of a single player; the formal definition is given in Section 3.1. It turns out that these have a structure very similar to that of Nash equilibria. In our example, local optima are given by $B_{SO} + \vec{\delta} \cdot \mathbb{N}$, thus with the same vector $\vec{\delta}$ as above, and with $B_{SO} = \{(0, 0), (1, 0), (2, 0), (2, 1)\}$.

The particular structures of these semilinear sets we obtain allow us to compute the prices of anarchy and stability for any given number of players. In fact, given n , one can easily find, in a set $B + \vec{\delta} \cdot \mathbb{N}$, all strategy profiles with n players. So one can compute the worst and the

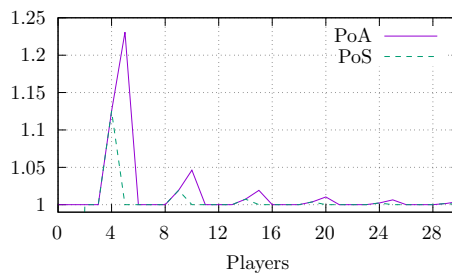


Figure 2 Prices of anarchy and stability as a function of n .

best Nash equilibria, as well as the social optimum for any given n . The plot in Fig. 2 shows how PoA and PoS evolve as n increases, and was calculated from the previous representations. This plot illustrates our objective of analyzing the inefficiency of these games precisely for varying n . Even in this simple example, PoA and PoS can significantly vary depending on n , and they are far from the known tight bounds for the whole class of networks. One can notice in Fig. 2 that both PoA and PoS seem to converge to 1 as n increases. This is indeed the case and is a consequence of our results (Theorem 14). We believe that this approach can allow one to better understand the specific network under analysis. Section 5 contains more examples.

Paper Overview. We provide formal definitions in Section 2. Section 3 characterizes the form of semilinear representations of local social optima; and Section 4 proves that of Nash equilibria. Section 5 shows how to use these representations to compute PoA and PoS. We provide more discussion on related work in Section 6, and present conclusions in Section 7.

2 Preliminaries

2.1 Network Congestion Games and Series-Parallel Arenas

A *network* is a weighted graph $\mathcal{A} = \langle V, E, \text{orig}, \text{dest}, \text{wgt}, \text{src}, \text{tgt} \rangle$, where V is a finite set of *vertices*, E is a finite set of *edges*, $\text{orig}: E \rightarrow V$ and $\text{dest}: E \rightarrow V$ indicate the origin and destination of each edge, $\text{wgt}: E \rightarrow \mathbb{N}_{>0}$ is a *weight function* assigning *positive* weights to edges, and src and tgt are, respectively, a *source* and a *target* states. Let $\text{In}(v)$ and $\text{Out}(v)$ denote, respectively, the set of incoming and outgoing edges of v . We restrict to acyclic networks in which all vertices are reachable from src and tgt is reachable from all vertices.

A *path* π of \mathcal{A} is a sequence $e_1 e_2 \dots e_n$ of edges with $\text{dest}(e_i) = \text{orig}(e_{i+1})$ for all $1 \leq i \leq n-1$. For an edge e and a path $\pi = e_1 e_2 \dots e_n$, we write $e \in \pi$ if $e = e_i$ for some $1 \leq i \leq |\pi|$. We will sometimes see paths as sets of edges and apply set operations such as intersection and set difference. Let $\text{Paths}_{\mathcal{A}}(s, t)$ denote the set of simple paths from s to t in \mathcal{A} , and let $\text{Paths}_{\mathcal{A}} = \text{Paths}_{\mathcal{A}}(\text{src}, \text{tgt})$.

In this work, we consider *series-parallel networks* [34]. These are built inductively from single edges using *serial* and *parallel* composition. Two networks $\mathcal{A}_1 = \langle V_1, E_1, \text{orig}_1, \text{dest}_1, \text{wgt}_1, \text{src}_1, \text{tgt}_1 \rangle$ and $\mathcal{A}_2 = \langle V_2, E_2, \text{orig}_2, \text{dest}_2, \text{wgt}_2, \text{src}_2, \text{tgt}_2 \rangle$ are *composed in series* to a new network denoted by $\mathcal{A}_1; \mathcal{A}_2 = \langle V, E, \text{orig}, \text{dest}, \text{wgt}, \text{src}, \text{tgt} \rangle$ obtained by taking the disjoint union of \mathcal{A}_1 and \mathcal{A}_2 , and merging the vertices tgt_1 and src_2 , and setting $\text{src} = \text{src}_1$, $\text{tgt} = \text{tgt}_2$. Two networks $\mathcal{A}_1 = \langle V_1, E_1, \text{orig}_1, \text{dest}_1, \text{wgt}_1, \text{src}_1, \text{tgt}_1 \rangle$ and $\mathcal{A}_2 = \langle V_2, E_2, \text{orig}_2, \text{dest}_2, \text{wgt}_2, \text{src}_2, \text{tgt}_2 \rangle$ are *composed in parallel* to a new network denoted by $\mathcal{A}_1 \parallel \mathcal{A}_2 = \langle V, E, \text{orig}, \text{dest}, \text{wgt}, \text{src}, \text{tgt} \rangle$ obtained by taking the disjoint union of \mathcal{A}_1 and \mathcal{A}_2 , and merging src_1 and src_2 , then merging tgt_1 and tgt_2 , and setting $\text{src} = \text{src}_1$, $\text{tgt} = \text{tgt}_1$. A network is said to be *series-parallel* if it is either a single edge, or it is a serial or parallel composition of series-parallel graphs.

A *network congestion game (NCG)* is a pair $\mathcal{G} = \langle \mathcal{A}, n \rangle$ where $\mathcal{A} = \langle V, E, \text{orig}, \text{dest}, \text{wgt}, \text{src}, \text{tgt} \rangle$ is a network, and $n \in \mathbb{N}$ is the number of players in the game. We consider the symmetric case where all players start at src and want to reach tgt . A *strategy* of a player is a path in $\text{Paths}_{\mathcal{A}}$. The setting can be seen as a one-shot game in which each player selects a strategy simultaneously. In our study, we do not need to identify players, we thus represent strategy profiles by counting how many players choose each strategy. That is, a *strategy profile* is a tuple $\vec{p} = (p_{\pi})_{\pi \in \text{Paths}_{\mathcal{A}}}$ where p_{π} is the number of players taking path π . In this case, the number of players is given by $\|\vec{p}\| = \sum_{\pi \in \text{Paths}_{\mathcal{A}}} p_{\pi}$; thus \vec{p} is a strategy profile in

the game $\langle \mathcal{A}, \|\vec{p}\| \rangle$. Let $\mathfrak{S}(\mathcal{A})$ denote the set of all strategy profiles, and $\mathfrak{S}_n(\mathcal{A})$ the set of strategy profiles with n players, that is, $\{\vec{p} \in \mathfrak{S}(\mathcal{A}) \mid n = \|\vec{p}\|\}$. For $\pi \in \text{Paths}_{\mathcal{A}}$, let $\vec{p} + \pi$ (resp. $\vec{p} - \pi$) denote the strategy profile obtained by incrementing (resp. decrementing) p_π by one.

Another useful notion we use is the *flow* of a strategy profile, which consists in the projection of a strategy profile to edges. Formally, given a strategy profile \vec{p} , $\text{flow}(\vec{p}) = (q_e)_{e \in E}$ where $q_e = \sum_{\pi \in \text{Paths}_{\mathcal{A}}: e \in \pi} p_\pi$, that is, q_e is the number of players that use the edge e in the profile \vec{p} . This vector satisfies the following flow equations:

$$\forall v \in V \setminus \{\text{src}, \text{tgt}\}, \quad \sum_{e \in \text{In}(v)} q_e = \sum_{e \in \text{Out}(v)} q_e. \quad (1)$$

We refer to a vector $\vec{q} = (q_e)_{e \in E}$ with nonnegative coefficients satisfying (1) as a *flow*; and denote by $\mathcal{F}(\mathcal{A})$ the set of all flows. Observe that $\mathcal{F}(\mathcal{A})$ is the image of $\mathfrak{S}(\mathcal{A})$ by flow . For a flow \vec{q} , let $\|\vec{q}\| = \sum_{e \in E: \text{orig}(e) = \text{src}} q_e$, which is the number of players. Let $\mathcal{F}_n(\mathcal{A})$ define the set of flows with n players as follows: $\mathcal{F}_n(\mathcal{A}) = \{\vec{q} \in \mathcal{F} \mid n = \|\vec{q}\|\}$. Observe that this corresponds to a flow of size n , and that several strategy profiles can project to the same flow.

For a strategy profile \vec{p} and $\pi \in \text{Paths}_{\mathcal{A}}$, each player using the path π incurs a cost equal to $\text{cost}_\pi(\vec{p}) = \sum_{e \in \pi} \text{wgt}(e) \cdot \text{flow}_e(\vec{p})$, where $\text{flow}_e(\vec{p})$ is the number of players using edge e in the strategy profile \vec{p} . The *social cost* of a strategy profile \vec{p} is the sum of the costs for all players, i.e., $\text{soccost}(\vec{p}) = \sum_{\pi \in \text{Paths}_{\mathcal{A}}} p_\pi \cdot \text{cost}_\pi(\vec{p})$. The *social optimum* of the game $\mathcal{G} = \langle \mathcal{A}, n \rangle$ is $\text{opt}(\mathcal{G}) = \min_{\vec{p} \in \mathfrak{S}_n(\mathcal{A})} \text{soccost}(\vec{p})$. A strategy profile $\vec{p} \in \mathfrak{S}_n$ in a game $\mathcal{G} = \langle \mathcal{A}, n \rangle$ is *socially optimal* if $\text{soccost}(\vec{p}) = \text{opt}(\mathcal{G})$.

Observe that the cost of a path in a strategy profile, and the social cost of a strategy profile, are determined by the flow of that profile. Thus, we define the social cost of a flow \vec{q} as $\text{soccost}(\vec{q}) = \sum_{e \in E} q_e^2 \cdot \text{wgt}(e)$ (in fact, q_e players use strategies that include e , and each of them pays $q_e \text{wgt}(e)$ for crossing this edge). A flow $\vec{q} \in \mathcal{F}_n$ is socially optimal if $\text{soccost}(\vec{q}) = \text{opt}(\mathcal{G})$.

A strategy profile \vec{p} is a Nash equilibrium if no player can reduce their cost by unilaterally changing strategy, i.e., if

$$\forall \pi \in \text{Paths}_{\mathcal{A}}, p_\pi > 0 \rightarrow \forall \pi' \in \text{Paths} \setminus \{\pi\}, \text{cost}_\pi(\vec{p}) \leq \text{cost}_{\pi'}(\vec{p}'), \quad (2)$$

where \vec{p}' is defined by $p'_\pi = p_\pi - 1$, $p'_{\pi'} = p_{\pi'} + 1$, and $p'_\tau = p_\tau$ for all other paths τ . In fact, $\text{cost}_\pi(\vec{p})$ is the cost of a player playing π in the profile \vec{p} , while $\text{cost}_{\pi'}(\vec{p}')$ is their cost in the new profile \vec{p}' obtained by switching from π to π' .

Let $\text{NE}(\mathcal{A})$ denote the set of strategy profiles satisfying (2), that is, the set of Nash equilibria, and let $\text{NE}_n(\mathcal{A})$ denote the set of Nash equilibria for n players. The *price of anarchy* is the ratio of the social cost of the worst Nash equilibrium, and the social optimum: $\text{PoA}(\langle \mathcal{A}, n \rangle) = \max_{\vec{p} \in \text{NE}_n} \text{soccost}(\vec{p}) / \text{opt}(\langle \mathcal{A}, n \rangle)$. The *price of stability* is the ratio of the social cost of the best Nash equilibrium, and the social optimum: $\text{PoS}(\langle \mathcal{A}, n \rangle) = \min_{\vec{p} \in \text{NE}_n} \text{soccost}(\vec{p}) / \text{opt}(\langle \mathcal{A}, n \rangle)$.

2.2 Presburger Arithmetic and Semilinear Sets

We recall the definition and some basic properties of semilinear sets; see e.g. [23] for more details. A set $S \subseteq \mathbb{N}^m$ is called *linear* if there is a *base vector* $\vec{b} \in \mathbb{N}^m$ and a finite set of *period vectors* $P = \{\vec{\delta}_1, \vec{\delta}_2, \dots, \vec{\delta}_p\}$ such that $S = \vec{b} + \vec{\delta}_1 \cdot \mathbb{N} + \vec{\delta}_2 \cdot \mathbb{N} + \dots + \vec{\delta}_p \cdot \mathbb{N}$, that is $S = \{\vec{b} + \lambda_1 \vec{\delta}_1 + \dots + \lambda_p \vec{\delta}_p \mid \lambda_1, \dots, \lambda_p \in \mathbb{N}\}$. Such a linear set S will be denoted as $L(\vec{b}, P)$.

A set $S \subseteq \mathbb{N}^m$ is said to be *semi-linear* if it is a finite union of linear sets. Therefore, a semi-linear set S can be written in the form $S = \cup_{i \in I} L(\vec{b}_i, P_i)$ where I is a finite set, P_i 's are finite sets of period vectors, and the \vec{b}_i are the base vectors of the same dimension.

Note that in a linear set $L(\vec{b}, P)$, P can be empty, which corresponds to a singleton set. Thus, finite sets are semilinear; and the union of any semilinear set with a finite set is semilinear. Furthermore, each semilinear set admits a *non-ambiguous* representation in the sense that $S = \cup_{i \in I} L(\vec{b}_i, P_i)$ such that each P_i is linearly independent and $L(\vec{b}_i, P_i) \cap L(\vec{b}_j, P_j) = \emptyset$ for all $i \neq j \in I$ [16, 25].

Presburger arithmetic is the first-order theory of integers without multiplication. It is well-known that any set expressible in Presburger arithmetic is *semilinear* [22]. So, in order to show that a set is semilinear, one can either exhibit its semilinear representation, or show that it is expressible in Presburger arithmetic.

3 Local Social Optima

In this section, our goal is to obtain a representation of social optima in a given network congestion game as a function of the number n of players. Characterizing the social optimum directly by a formula brings two difficulties. First, expressing that a flow \vec{q} is optimal would require to quantify over all flows \vec{q}' and writing that $\text{soccost}(\vec{q}) \leq \text{soccost}(\vec{q}')$, so such a formula contains universal quantifiers. Second, the formula is quadratic since $\text{soccost}(\vec{q}) = \sum_{e \in E} q_e^2 \cdot \text{wgt}(e)$, so this cannot be represented by a semilinear set.

Here, we introduce the notion of *local optimality* which allows us to circumvent both difficulties, providing semilinear representations which, moreover, allow us to compute the global optimum.

3.1 Locally-Optimal Profiles

Let us fix a series-parallel network \mathcal{A} . Intuitively, a strategy profile is *locally-optimal* if the social cost cannot be reduced by exchanging one path for another. Formally, $\vec{p} \in \mathfrak{S}_n(\mathcal{A})$ is locally-optimal if for all $\pi, \pi' \in \text{Paths}_{\mathcal{A}}$ with $p_\pi > 0$, $\text{soccost}(\vec{p}) \leq \text{soccost}(\vec{p} - \pi + \pi')$. By extension, a flow $\vec{q} \in \mathcal{F}_n(\mathcal{A})$ is locally-optimal if it is the image of a locally-optimal strategy profile. Observe that the (global) social optimum is locally-optimal.

In the following lemma, we see paths π, π' as sets of edges.

► **Lemma 1.** *In a network congestion game $\langle \mathcal{A}, n \rangle$, a flow \vec{q} is locally-optimal if, and only if, for all $\pi, \pi' \in \text{Paths}_{\mathcal{A}}$ such that $\forall e \in \pi, q_e > 0$,*

$$\sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot (2q_e - 1) \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (2q_e + 1). \quad (3)$$

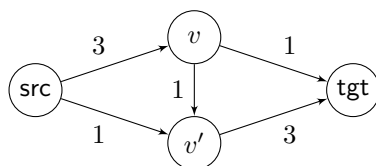
We define $\text{LocOpt}(\mathcal{A})$ to be the set of locally-optimal flows, and $\text{LocOpt}_n(\mathcal{A})$ those with n players. It follows from Lemma 1 that $\text{LocOpt}(\mathcal{A})$ and $\text{LocOpt}_n(\mathcal{A})$ are expressible in Presburger arithmetic, and are thus semilinear. We will now characterize the form of the semilinear set describing $\text{LocOpt}(\mathcal{A})$ by proving that it admits a single and computable period vector $\vec{\delta}$, that is, $\text{LocOpt}(\mathcal{A})$ can be written as $B \cup \bigcup_{i \in I} L(\vec{b}_i, \vec{\delta})$, where B is a finite set of flows, I is a finite set of indices, and $(\vec{b}_i)_{i \in I}$ are the base vectors.

3.2 Large Numbers of Players

To simplify the proof of the characterization of the period vector $\vec{\delta}$, we would like to consider instances in which (3) holds for *all* paths $\pi, \pi' \in \text{Paths}_{\mathcal{A}}$, that is, we would like to get rid of the assumption on π in Lemma 1. It turns out that the assumption that $q_e > 0$ for all edges e

of π holds whenever the number of players is large enough. Moreover, we do not lose generality by focusing on these instances; in fact, as we will see, a semilinear representation with the same period vector \vec{d} for *all* locally-optimal profiles can be derived once this representation is established for instances with large numbers of players.

The next lemma shows that all edges are used in locally-optimal profiles whenever the number of players is sufficiently large. This property is specific to series-parallel graphs and may not hold in a more general network, as the following example shows.



■ **Figure 3** A network with an edge $v \rightarrow v'$ that is never used in any locally-optimal profile.

► **Example 2.** Consider the network of Fig. 3. We claim that the edge from v to v' cannot be used in any locally-optimal profile. Write a , b and c for the number of players taking paths $\pi_a : \text{src} \rightarrow v \rightarrow \text{tgt}$, $\pi_b : \text{src} \rightarrow v \rightarrow v' \rightarrow \text{tgt}$ and $\pi_c : \text{src} \rightarrow v' \rightarrow \text{tgt}$, respectively. Observe (by contradiction) that if there are at least two players, then paths π_a and π_c will be taken by at least one player. Writing Eq. (3) for π_a and π_c yields $-1 \leq a - c \leq 1$. Assuming $b > 0$, we can also apply Eq. (3) to π_b and π_a , and get $-2a + 8b + 6c - 5 \leq 0$. It follows $8b + 4c - 7 \leq 0$. This cannot be preserved when the number of players grows. Hence π_b cannot be used in any locally-optimal profile. ◀

► **Lemma 3.** *In all series-parallel networks \mathcal{A} , there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, all flows $\vec{q} \in \text{LocOpt}_n(\mathcal{A})$ are such that $q_e > 0$ for all $e \in E$.*

Proof. We inductively define a value $n_0(\mathcal{A})$ for each series-parallel network \mathcal{A} , such that $n_0(\mathcal{A}) \geq 4$ (for technical reasons) and satisfying the following property:

$$\forall k \geq 1, \forall n \geq k \cdot n_0(\mathcal{A}), \forall \vec{q} \in \text{LocOpt}_n(\mathcal{A}), \forall e \in E, q_e \geq k.$$

The lemma follows by taking $k = 1$.

If \mathcal{A} is a single edge, we define $n_0(\mathcal{A}) = 4$, and the property trivially holds.

If $\mathcal{A} = \mathcal{A}_1; \mathcal{A}_2$, then we define $n_0(\mathcal{A}) = \max(n_0(\mathcal{A}_1), n_0(\mathcal{A}_2))$. Observe that if \vec{q} is locally-optimal in \mathcal{A} , then each $\vec{q}|_{\mathcal{A}_i}$ is locally-optimal in \mathcal{A}_i , and they have the same number of players. So the property holds by induction.

The non-trivial case is when $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$. Let $m = \max(n_0(\mathcal{A}_1), n_0(\mathcal{A}_2))$, and $n_0(\mathcal{A}) = 2|E|Wm^2$, where E is set of edges of \mathcal{A} and W is the largest weight of \mathcal{A} (notice that $W \geq 1$). Then $m \geq 4$ and $n_0(\mathcal{A}) \geq 4$. Take $n \geq kn_0(\mathcal{A})$, and $\vec{q} \in \text{LocOpt}_n$.

We show that for all $i \in \{1, 2\}$, we have $\|\vec{q}|_{\mathcal{A}_i}\| \geq kn_0(\mathcal{A}_i)$. Towards a contradiction, assume for example that $\|\vec{q}|_{\mathcal{A}_1}\| < kn_0(\mathcal{A}_1)$. Then $\|\vec{q}|_{\mathcal{A}_2}\| > n - kn_0(\mathcal{A}_1) \geq 2k|E|W(m^2 - m)$ (because $kn_0(\mathcal{A}_1) \leq km \leq 2k|E|Wm$).

For all pair $(\pi_1, \pi_2) \in \text{Paths}_{\mathcal{A}_1} \times \text{Paths}_{\mathcal{A}_2}$, by (3) we have that $\sum_{e \in \pi_2} \text{wgt}(e)(2q_e - 1) \leq \sum_{e \in \pi_1} \text{wgt}(e)(2q_e + 1)$. Take an arbitrary edge e_0 of π_2 ; then e_0 does not appear in π_1 , and we get

$$2q_{e_0} \leq 2W \sum_{e \in \pi_1} q_e + \sum_{e \in \pi_2} \text{wgt}(e) + \sum_{e \in \pi_1} \text{wgt}(e) < 2W|E| \cdot kn_0(\mathcal{A}_1) + 2W|E| \leq 2W|E|(km + 1).$$

Now, take $k' = 2k|E|W(m-1)$. Then $k' \geq 1$, and $\|\vec{q}|_{\mathcal{A}_2}\| > k'm \geq kn_0(\mathcal{A}_2)$. By induction hypothesis applied to \mathcal{A}_2 , we have $q_{e_0} \geq k'$, which implies

$$2k|E|W(m-1) \leq W|E|(km+1)$$

hence $m \leq 2 + \frac{1}{k} \leq 3$, which is a contradiction since $m \geq 4$. \blacktriangleleft

Let $\text{LocOpt}_{\geq n_0}(\mathcal{A}) = \bigcup_{n \geq n_0} \text{LocOpt}_n(\mathcal{A})$. By the previous lemma, all $\vec{q} \in \text{LocOpt}_{\geq n_0}(\mathcal{A})$ satisfy (3) for all pairs of paths π, π' . Observe that $\text{LocOpt}(\mathcal{A})$ and $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ have the same period vectors. In fact, $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ differs from $\text{LocOpt}(\mathcal{A})$ only by a finite set; so, given a semilinear representation $\bigcup_{i \in I} L(\vec{b}_i, P_i)$ for the former, one can obtain a representation for the latter as $B \cup \bigcup_{i \in I} L(\vec{b}_i, P_i)$ where B is the finite difference between the two. Therefore, establishing that $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ has a single period vector suffices to prove the same result for $\text{LocOpt}(\mathcal{A})$. Note also that I is non-empty here since the set $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ is infinite.

The following lemma shows that the period vectors of $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ assign the same cost to all paths. Intuitively, this is because if the cost along two paths were different in the period vector, then by adding a large number of copies of the period vector to its base vector, one could amplify this difference and obtain a vector that is not locally-optimal.

► **Lemma 4.** *In a series-parallel network \mathcal{A} , for all period vectors $\vec{d} \in \mathbb{N}^E$ of a semilinear representation of $\text{LocOpt}_{\geq n_0}(\mathcal{A})$, there exists $\kappa \geq 0$ such that for all $\pi \in \text{Paths}_{\mathcal{A}}$, we have $\sum_{e \in \pi} \text{wgt}(e) \cdot d_e = \kappa$.*

3.3 A Unique Period Vector: The Characteristic Vector

We now establish that $\text{LocOpt}(\mathcal{A})$ admits a unique period vector. For any $\kappa \in \mathbb{R}$, we study the following system $\mathcal{E}(\kappa)$ of equations with unknowns $\{q_e\}_{e \in E}$:

$$\forall \pi \in \text{Paths}_{\mathcal{A}}, \sum_{e \in \pi} \text{wgt}(e) \cdot q_e = \kappa, \quad (4)$$

$$\forall v \in V \setminus \{\text{src}, \text{tgt}\}, \sum_{e \in \text{In}(v)} q_e - \sum_{e \in \text{Out}(v)} q_e = 0. \quad (5)$$

Note that all period vectors of $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ satisfy the above. In fact, (4) comes from Lemma 4, and (5) is the set of flow equations (1).

The following lemma states that $\mathcal{E}(\kappa)$ has a unique solution whenever we fix κ .

► **Lemma 5.** *For a series-parallel network \mathcal{A} , for each $\kappa \in \mathbb{R}$, the system $\mathcal{E}(\kappa)$ admits a unique solution.*

The system $\mathcal{E}(\kappa)$ is actually the characterization of the flows of Nash equilibria in the non-atomic congestion games, and the unicity of the solution of this equation system is known (see [12]). We represent the system $\mathcal{E}(\kappa)$ in the matrix form as $M_{\mathcal{A}} \cdot X = \kappa \vec{b}$, where $X = (q_e)_{e \in E}$ is the vector of unknowns, and \vec{b} a $\{0, 1\}$ -column vector. Lemma 5 means that $M_{\mathcal{A}}$ admits a left-inverse $M_{\mathcal{A}}^{-1}$. It follows that all period vectors can be written as $\kappa M_{\mathcal{A}}^{-1} \vec{b}$ for some κ . Since $M_{\mathcal{A}}$ and \vec{b} have integer coefficients, $M_{\mathcal{A}}^{-1} \vec{b}$ is a vector with rational coefficients.

► **Definition 6 (Characteristic Vector).** *Let κ_0 denote the least rational number such that $\kappa_0 M_{\mathcal{A}}^{-1} \vec{b}$ is integer. We define $\vec{\delta} = \kappa_0 M_{\mathcal{A}}^{-1} \vec{b}$, called the characteristic vector of \mathcal{A} .*

Since period vectors of $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ have natural number coefficients, any κ corresponding to a period vector is also a natural. We have that all period vectors are integer multiples of $\vec{\delta}$. In fact, if $\vec{d} = \kappa M_{\mathcal{A}}^{-1} \vec{b}$ is a period vector, then $\vec{d} = \frac{\kappa}{\kappa_0} \vec{\delta}$, and $\frac{\kappa}{\kappa_0}$ is an integer since otherwise its denominator would divide $\vec{\delta}$, which would contradict the minimality of κ_0 .

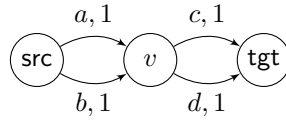
► **Corollary 7.** *Consider a series-parallel network \mathcal{A} , and its characteristic vector $\vec{\delta}$. There exist finite sets of vectors B and $\{\vec{b}_i \mid i \in I\}$ such that $\text{LocOpt}(\mathcal{A}) = B \cup \bigcup_{i \in I} L(\vec{b}_i, \vec{\delta})$.*

Proof. Since each linear set can be assumed to have linearly-independent period vectors (Section 2.2) all linear sets included in $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ can be written in the form of $\vec{b} + m\vec{\delta}\mathbb{N}$,

We show that $m = 1$. By Lemma 3, \vec{b} has only positive coefficients, so it satisfies (3) for all pairs of paths. We show that $\vec{b} + k\vec{\delta}$ also satisfies (3) for all $k \geq 0$, which proves that $L(\vec{b}, \vec{\delta})$ is included in $\text{LocOpt}_{\geq n_0}(\mathcal{A})$. For all paths π, π' , consider (3) by adding an identical term to both sides:

$$\begin{aligned} \sum_{e \in \pi \setminus \pi'} \text{wgt}(e)(2q_e - 1) + 2k \sum_{e \in \pi} \text{wgt}(e)\delta_e &\leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (2q_e + 1) + 2k \sum_{e \in \pi'} \text{wgt}(e)\delta_e \\ \sum_{e \in \pi \setminus \pi'} \text{wgt}(e)(2q_e - 1) + 2k \sum_{e \in \pi \setminus \pi'} \text{wgt}(e)\delta_e &\leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (2q_e + 1) + 2k \sum_{e \in \pi' \setminus \pi} \text{wgt}(e)\delta_e \\ \sum_{e \in \pi \setminus \pi'} \text{wgt}(e)(2(q_e + k\delta_e) - 1) &\leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (2(q_e + k\delta_e) + 1). \end{aligned}$$

Therefore, $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ can be written in the form $\bigcup_{i \in I} L(\vec{b}_i, \vec{\delta})$; and since $\text{LocOpt}(\mathcal{A})$ differs from $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ by a finite set, it can be represented by $B \cup \bigcup_{i \in I} L(\vec{b}_i, \vec{\delta})$ where $B = \text{LocOpt}(\mathcal{A}) \setminus \text{LocOpt}_{\geq n_0}(\mathcal{A})$. ◀



■ **Figure 4** Network whose set of local optima admit several period vectors.

► **Remark 8.** Note that we chose here to study the semilinear representations of locally-optimal flows, rather than locally-optimal strategy profiles. The latter are also expressible in Presburger arithmetic, thus also admit a semilinear representation. However, the set of locally-optimal strategy profiles admit, in general, several linearly independent period vectors, so their representation is more complex, and more difficult to use, for instance, to compute the global optimum for given number n of players.

To see this, consider the example of Fig. 4. Consider the strategy profile \vec{p} with $p_{ac} = p_{bd} = 1$ and $p_{ad} = p_{bc} = 0$; and \vec{p}' such that $p'_{ac} = p'_{bd} = 0$ and $p'_{ad} = p'_{bc} = 1$. For all $k \geq 0$, both $k\vec{p}$ and $k\vec{p}'$ are socially optimal, but they are linearly independent. In larger networks, there can be a larger number of period vectors due to similar phenomena. Notice however that $\text{flow}(\vec{p}) = \text{flow}(\vec{p}')$, that is, the projections of these period vectors to their flows are identical, and are, in fact, equal to the characteristic vector.

4 Nash Equilibria

In this section, we show how to compute a semilinear representation of the flows of Nash equilibria, which will allow us to compute the costs of the best and the worst equilibria.

32:10 Semilinear Representations for Series-Parallel Atomic Congestion Games

The following lemma is a characterization of Nash equilibria that follows from (2).

► **Lemma 9.** *Given a network \mathcal{A} , a strategy profile \vec{p} is a Nash equilibrium if, and only if,*

$$\forall \pi, \pi' \in \text{Paths}, p_\pi > 0 \implies \sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot q_e \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (q_e + 1), \quad (6)$$

where $\vec{q} = \text{flow}(\vec{p})$.

It follows from the previous lemma that Nash equilibria, but also their flows, are definable in Presburger arithmetic.

► **Lemma 10.** *The sets $\text{NE}(\mathcal{A})$ and $\text{flow}(\text{NE}(\mathcal{A}))$ are semilinear.*

The study of the semilinear representation is similar to what was done for locally-optimal profiles. We prove that the period vectors of $\text{flow}(\text{NE}(\mathcal{A}))$ are colinear to $\vec{\delta}$, thus establishing the form of their semilinear representation. In particular, we use Lemma 5 and show that the period vectors are multiples of the characteristic vector. Here, we consider $\text{flow}(\text{NE}(\mathcal{A}))$ rather than $\text{NE}(\mathcal{A})$ due to Remark 8 which also holds for Nash equilibria.

The following lemma shows that a semilinear representation of $\text{flow}(\text{NE}(\mathcal{A}))$ can be deduced from that of $\text{NE}(\mathcal{A})$ and follows by the definition of flow . In particular, all period vectors of the former are projections of the period vectors of the latter by flow .

► **Lemma 11.** *For any semilinear representation $B \cup \bigcup_{i \in I} L(\vec{b}_i, P_i)$ of $\text{NE}(\mathcal{A})$, we have $\text{flow}(\text{NE}(\mathcal{A})) = \text{flow}(B) \cup \bigcup_{i \in I} L(\text{flow}(\vec{b}_i), \text{flow}(P_i))$, where $\text{flow}(X) = \{\text{flow}(\vec{p}) \mid \vec{p} \in X\}$.*

As in the case of locally-optimal strategy profiles, we establish that the period vectors of $\text{flow}(\text{NE}(\mathcal{A}))$ for series-parallel networks have constant cost along all paths.

► **Lemma 12.** *For a series-parallel network \mathcal{A} , and any period vector \vec{q} of $\text{flow}(\text{NE}(\mathcal{A}))$, there exists κ , such that for all $\pi \in \text{Paths}_{\mathcal{A}}$, $\sum_{e \in \pi} \text{wgt}(e) \cdot q_e = \kappa$.*

Proof. We use structural induction on the series-parallel network. The base case is when the network is a single edge, which trivially satisfies the property.

Consider a network $\mathcal{A} = \langle V, E, \text{orig}, \text{dest}, \text{wgt}, s, t \rangle$ with $\mathcal{A} = \mathcal{A}_1; \mathcal{A}_2$ with $\mathcal{A}_1 = \langle V_1, E_1, \text{orig}_1, \text{dest}_1, \text{wgt}_1, s_1, t_1 \rangle$ and $\mathcal{A}_2 = \langle V_2, E_2, \text{orig}_2, \text{dest}_2, \text{wgt}_2, s_2, t_2 \rangle$.

Let \vec{q} be a period vector of $\text{flow}(\text{NE}(\mathcal{A}))$. By Lemma 11, there exists a period vector \vec{p} of $\text{NE}(\mathcal{A})$, with $\vec{q} = \text{flow}(\vec{p})$. Let us show that each $\vec{q}|_{\mathcal{A}_i}$ is a period vector of $\text{flow}(\text{NE}(\mathcal{A}_i))$. It suffices to show that $\text{prj}_{\mathcal{A}_i}(\vec{p})$ is a period vector of $\text{NE}(\mathcal{A}_i)$: since $\vec{q}|_{\mathcal{A}_i} = \text{flow}(\text{prj}_{\mathcal{A}_i}(\vec{p}))$, we can then conclude by Lemma 11.

For some base vector \vec{b} , we have that $\vec{b} + k\vec{p} \in \text{NE}(\mathcal{A})$ for all $k \geq 0$. By Lemma 16, $\text{prj}_{\mathcal{A}_i}(\vec{b} + k\vec{p}) = \text{prj}_{\mathcal{A}_i}(\vec{b}) + k \cdot \text{prj}_{\mathcal{A}_i}(\vec{p}) \in \text{NE}(\mathcal{A}_i)$, thus $\text{prj}_{\mathcal{A}_i}(\vec{p})$ is indeed a period vector.

Let us call $\vec{q}^i = \vec{q}|_{\mathcal{A}_i}$. By the induction hypothesis, there exist constants κ_1, κ_2 such that

$$\forall \pi \in \text{Paths}_{\mathcal{A}_i}, \sum_{e \in \pi} \text{wgt}(e) \cdot q_e^i = \kappa_i.$$

It follows that

$$\forall \pi \in \text{Paths}_{\mathcal{A}}, \sum_{e \in \pi} \text{wgt}(e) \cdot q_e = \sum_{e \in \pi_1} \text{wgt}(e) \cdot q_e^1 + \sum_{e \in \pi_2} \text{wgt}(e) \cdot q_e^2 = \kappa_1 + \kappa_2,$$

where $\pi_1 \pi_2$ denotes the decomposition of π such that $\pi_i \in \text{Paths}_{\mathcal{A}_i}$.

Consider now the case $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$. Let \vec{q} be a period vector of $\text{flow}(\text{NE}(\mathcal{A}))$. By Lemma 11, there exists a period vector \vec{p} of $\text{NE}(\mathcal{A})$, with $\vec{q} = \text{flow}(\vec{p})$. Let us show that each $\vec{q}|_{\mathcal{A}_i}$ is a period vector of $\text{flow}(\text{NE}(\mathcal{A}_i))$. The argument is identical to the first case, using Lemma 17 in place of Lemma 16. It suffices to show that $\vec{p}|_{\mathcal{A}_i}$ is a period vector of $\text{NE}(\mathcal{A}_i)$ since we can then conclude by Lemma 11. For some base vector \vec{b} , we have that $\vec{b} + k\vec{p} \in \text{NE}(\mathcal{A})$ for all $k \geq 0$. By Lemma 17, $(\vec{b} + k\vec{p})|_{\mathcal{A}_i} \in \text{NE}(\mathcal{A}_i)$, and since $(\vec{b} + k\vec{p}) = \vec{b}|_{\mathcal{A}_i} + k\vec{p}|_{\mathcal{A}_i}$, $\vec{p}|_{\mathcal{A}_i}$ is indeed a period vector.

By the induction hypothesis, there exist κ_1, κ_2 such that

$$\forall \pi \in \text{Paths}_{\mathcal{A}_i}, \sum_{e \in \pi} \text{wgt}(e) \cdot q_e = \kappa_i.$$

We need to prove that $\kappa_1 = \kappa_2$. Assume otherwise; for instance, $\kappa_1 < \kappa_2$. Then, for paths $\pi_1, \pi_2 \in \text{Paths}_{\mathcal{A}}$, we have $\text{cost}_{\pi_1}(\vec{q}) < \text{cost}_{\pi_2}(\vec{q})$. Consider a period vector \vec{p} of $\text{NE}(\mathcal{A})$ with $\text{flow}(\vec{p}) = \vec{q}$. Observe that \vec{p} and its multiples are also a Nash equilibria (by Lemma 9). Then, there must exist $\pi_2 \in \text{Paths}_{\mathcal{A}_2}$ such that $p_{\pi_2} > 0$; if not, paths in $\text{Paths}_{\mathcal{A}_2}$ would become profitable deviations for $k\vec{p}$ for large k . Take such a $\pi_2 \in \text{Paths}_{\mathcal{A}_2}$. By (6), for all $\pi_1 \in \text{Paths}_{\mathcal{A}_1}$, and all $k \geq 0$, we have

$$\sum_{e \in \pi_2 \setminus \pi_1} \text{wgt}(e)kq_e \leq \sum_{e \in \pi_1 \setminus \pi_2} \text{wgt}(e)(kq_e + 1).$$

Since π_1 and π_2 are disjoint, we get

$$\sum_{e \in \pi_2} \text{wgt}(e)kq_e \leq \sum_{e \in \pi_1} \text{wgt}(e)(kq_e + 1),$$

hence $k(\kappa_2 - \kappa_1) \leq \sum_{e \in \pi_1} \text{wgt}(e)$, which is a contradiction for large k . This concludes the proof. \blacktriangleleft

It follows from Lemma 5 that all period vectors of $\text{flow}(\text{NE}(\mathcal{A}))$ are multiples of $\vec{\delta}$, the characteristic vector of \mathcal{A} .

► **Corollary 13.** *For a series-parallel network \mathcal{A} , the set $\text{flow}(\text{NE}(\mathcal{A}))$ admits a semilinear representation in the form $B \cup \bigcup_{i \in I} L(\vec{b}^i, m_i \vec{\delta})$, for a finite set B , and finitely-many base vectors \vec{b}^i and natural numbers m_i .*

We do not know whether the m_i in the above corollary can be different from 1. We did not encounter such a case in our experiments, and we conjecture that $\vec{\delta}$ is the only period vector of $\text{flow}(\text{NE}(\mathcal{A}))$.

An immediate consequence of Corollaries 7 and 13 is that the prices of anarchy and stability converge to 1 for a fixed series-parallel network, when the number of players goes to infinity. This is intuitively due to the fact that both $\text{LocOpt}(\mathcal{A})$ and $\text{flow}(\text{NE}(\mathcal{A}))$ have the same direction $\vec{\delta}$. This result already appeared recently in [37]; our setting provides an alternative proof.

► **Theorem 14.** *For series-parallel networks \mathcal{A} , $\lim_{n \rightarrow \infty} \text{PoA}(\langle \mathcal{A}, n \rangle) = \lim_{n \rightarrow \infty} \text{PoS}(\langle \mathcal{A}, n \rangle) = 1$.*

Proof. Consider $\text{worst}_n = \max_{\vec{p} \in \text{NE}_n(\mathcal{A})} \text{soccost}(\vec{p})$, $\text{best}_n = \min_{\vec{p} \in \text{NE}_n(\mathcal{A})} \text{soccost}(\vec{p})$, and $\text{opt}_n = \min_{\vec{p} \in \mathcal{F}_n(\mathcal{A})} \text{soccost}(\vec{p})$. We show that $\lim_{n \rightarrow \infty} \text{worst}_n / \text{opt}_n = \lim_{n \rightarrow \infty} \text{best}_n / \text{opt}_n = 1$.

As we already argued, the social cost only depends on the flow, so worst_n and best_n can be computed by maximizing or minimizing the social cost among $\text{flow}(\text{NE}(\mathcal{A}))$ restricted to n players. The optimum can be computed by minimizing over $\text{LocOpt}_n(\mathcal{A})$ since the global optimum is also locally optimal.

32:12 Semilinear Representations for Series-Parallel Atomic Congestion Games

Let $\text{flow}(\text{NE}(\mathcal{A})) = B \cup \bigcup_{i \in I} L(\vec{b}^i, m_i \vec{\delta})$, and $\text{LocOpt}(\mathcal{A}) = B' \cup \bigcup_{i \in I'} L(\vec{b}^i, \vec{\delta})$. Consider $n > \max_{\vec{b} \in B \cup B'} \|\vec{b}\|$, so that all Nash equilibria with n players belong to some $L(\vec{b}^i, m_i \vec{\delta})$, and similarly, all local optima with n players are in some $L(\vec{b}^i, \vec{\delta})$. Note that if a strategy profile \vec{p} belong to $L(\vec{b}^i, m_i \vec{\delta})$, there exists $k \in \mathbb{N}$ with $\vec{p} = \vec{b}^i + k m_i \vec{\delta}$, which implies that $\|\vec{p}\| - \|\vec{b}^i\| \equiv 0 \pmod{m_i \|\vec{\delta}\|}$, and $k = \frac{\|\vec{p}\| - \|\vec{b}^i\|}{m_i \|\vec{\delta}\|}$. We have that

$$\text{worst}_n = \max \left\{ \text{soccost} \left(\vec{b}^i + \vec{\delta} \cdot \frac{n - \|\vec{b}^i\|}{\|\vec{\delta}\|} \right) \mid i \in I, n - \|\vec{b}^i\| \equiv 0 \pmod{m_i \|\vec{\delta}\|} \right\}.$$

Similarly,

$$\text{opt}_n = \min \left\{ \text{soccost} \left(\vec{b}^i + \vec{\delta} \cdot \frac{n - \|\vec{b}^i\|}{\|\vec{\delta}\|} \right) \mid i \in I, n - \|\vec{b}^i\| \equiv 0 \pmod{\|\vec{\delta}\|} \right\}.$$

Consider $(i_0, i'_0) \in I \times I'$ such that worst_n is maximized for $i_0 \in I$, and opt_n is minimized for $i'_0 \in I'$ for infinitely many n . Let $(\alpha_k)_{k \geq 0}$ denote the increasing sequence of indices n such that this is the case. We are going to show that the limit of the sequence $(\text{worst}_{\alpha_k} / \text{opt}_{\alpha_k})_{k \in \mathbb{N}}$ is 1 (independently of i_0 and i'_0), which yields the result.

We have

$$\begin{aligned} \text{worst}_{\alpha_k} &= \text{soccost} \left(\vec{b}_{i_0} + \frac{\alpha_k - \|\vec{b}^i\|}{\|\vec{\delta}\|} \vec{\delta} \right) \\ &= \sum_{e \in E} \text{wgt}(e) \left(\frac{\delta_e}{\|\vec{\delta}\|} \right)^2 \alpha_k^2 + 2 \sum_{e \in E} \text{wgt}(e) \frac{\delta_e (b_e^i - \delta_e \|\vec{b}^i\| / \|\vec{\delta}\|)}{\|\vec{\delta}\|} \alpha_k \\ &\quad + \sum_{e \in E} \text{wgt}(e) \left(b_e^i - \delta_e \cdot \frac{\|\vec{b}^i\|}{\|\vec{\delta}\|} \right)^2. \end{aligned}$$

We have $\text{worst}_{\alpha_k} = A \alpha_k^2 + o(\alpha_k)$ where $A = \sum_{e \in E} \text{wgt}(e) \left(\frac{\delta_e}{\|\vec{\delta}\|} \right)^2$.

A similar expression can be obtained for opt_{α_k} since it has the same form as worst_{α_k} . In particular, the first term is again A . We can obtain that $\text{opt}_{\alpha_k} \geq A \alpha_k^2 - \frac{2E \|\vec{b}^i\|}{\|\vec{\delta}\|}$. Hence,

$$1 \leq \frac{\text{worst}_{\alpha_k}}{\text{opt}_{\alpha_k}} \leq \frac{A \alpha_k^2 + o(\alpha_k)}{A \alpha_k^2 + o(1)} = 1 + o\left(\frac{1}{\alpha_k}\right).$$

It follows that $\lim_{n \rightarrow \infty} \text{PoA}(\langle \mathcal{A}, n \rangle) = 1$.

This also implies that $\lim_{n \rightarrow \infty} \text{PoS}(\langle \mathcal{A}, n \rangle) = 1$. ◀

5 Computation of PoA and PoS

Semilinear Representations

Let us show how the semilinear representation for $\text{LocOpt}(\mathcal{A})$ can be computed. First, the characteristic vector $\vec{\delta}$ can be computed by solving the homogeneous equation system $\mathcal{E}(0)$ using a symbolic solver (so as to obtain a rational solution), and multiplying the unique solution by the gcd of its coefficients. Next, one can incrementally construct the semilinear representation using an integer-arithmetic solver to find the linear sets and the finite set B . This can be done with quantifier-free formulas only. Although integer linear programming is already NP-hard [6, 35], available solvers are efficient for small instances.

At a given iteration, assume that the current subset of $\text{LocOpt}(\mathcal{A})$ is $B \cup \bigcup_{i \in I} L(\vec{b}^i, \vec{\delta})$. We write a linear quantifier-free formula $\phi(\vec{q})$ with free variables a flow \vec{q} which requires that \vec{q} is locally optimal (by Lemma 1), and that \vec{q} is not included in the current set. We already saw that the former is a Presburger formula. The latter constraints can be written as $\bigwedge_{\vec{b} \in B} \vec{q} \neq \vec{b} \wedge \bigwedge_{i \in I} \vec{q} \notin L(\vec{b}^i, \vec{\delta})$. Here, $\vec{q} \notin L(\vec{b}^i, \vec{\delta}) \equiv \neg(\exists k. \vec{q} = \vec{b}^i + k\vec{\delta})$ but the existentially quantified k can be determined from the number of players of $\vec{q}, \vec{b}^i, \vec{\delta}$, so this can be simplified as follows:

$$(\exists k. \vec{q} = \vec{b}^i + k\vec{\delta}) \equiv (\|\vec{q}\| - \|\vec{b}^i\|) \equiv 0 \pmod{\|\vec{\delta}\|} \wedge \bigwedge_{e \in E} q_e = b_e^i + (\|\vec{q}\| - \|\vec{b}^i\|)\delta_e / \|\vec{\delta}\|,$$

where $\|\vec{\delta}\|$ and $\|\vec{b}^i\|$ are fixed numbers.

If $\phi(\vec{q})$ is not satisfiable, then the current representation is complete. Otherwise, a model satisfying the above formula gives a new vector \vec{q} that is locally optimal. To determine whether \vec{q} should belong to B , or whether $L(\vec{q}, \vec{\delta})$ is to be added to our set, we simply check if $\vec{q} + \vec{\delta}$ satisfies the condition of Lemma 1: if this is the case, then we keep the linear set, otherwise we add \vec{q} to B . In fact, for all $k \geq 1$, $\vec{q} + k\vec{\delta}$ has the same set of paths π satisfying $\forall e \in \pi, (q_e + k\delta_e) > 0$, so this check is sufficient. Since the set admits a finite semilinear representation, this procedure terminates.

Let us explain the computation of $\text{flow}(\text{NE}(\mathcal{A}))$. Let $\text{NE}(\vec{p})$ denote the linear constraints of (6). Assume that we currently have a subset of $\text{flow}(\text{NE}_{\mathcal{A}})$ in the form $B \cup \bigcup_{i \in I} L(\vec{b}^i, m_i \vec{\delta})$. The following formula $\psi(\vec{p}, \vec{q})$ with free variables \vec{p}, \vec{q} is satisfiable iff some Nash equilibrium \vec{p} (with $\text{flow}(\vec{p}) = \vec{q}$) is not in the set:

$$\psi(\vec{p}, \vec{q}) = \text{NE}(\vec{p}) \wedge \text{flow}(\vec{p}) = \vec{q} \wedge \bigwedge_{i \in I} \vec{q} \notin L(\vec{b}^i, m_i \vec{\delta}) \wedge \bigwedge_{\vec{b} \in B} \vec{q} \neq \vec{b}.$$

Assume that this is satisfiable, and let \vec{p}, \vec{q} be a model. We need to check whether \vec{q} is to be added to B , or whether $L(\vec{q}, m_i \vec{\delta})$, for some m_i , is to be included. We use the following properties of the period vectors of $\text{NE}(\mathcal{A})$. Let us first define $S_{\vec{\delta}} = \{\pi \in \text{Paths}_{\mathcal{A}} \mid \sum_{e \in \pi} \text{wgt}(e)\delta_e \leq \min_{\pi' \in \text{Paths}_{\mathcal{A}}} \sum_{e \in \pi'} \text{wgt}(e)\delta_e\}$. Intuitively, $S_{\vec{\delta}}$ is the set of paths in $\text{Paths}_{\mathcal{A}}$ with minimum cost in the profile $\vec{\delta}$.

► **Lemma 15.** *Let $\vec{p}, \vec{p}' \in \text{NE}(\mathcal{A})$ such that $\text{flow}(\vec{p}') = m\vec{\delta}$. We have $L(\vec{p}, \vec{p}') \subseteq \text{NE}(\mathcal{A})$ iff for all $\pi \in \text{Paths}_{\mathcal{A}}$, $\pi \notin S_{\vec{p}} \Rightarrow p'_\pi = 0$.*

Proof. Assume $L(\vec{p}, \vec{p}') \subseteq \text{NE}(\mathcal{A})$. Then, for all π such that $p_\pi > 0$ or $p'_\pi > 0$, we have

$$\begin{aligned} \forall \pi' \in \text{Paths}(\mathcal{A}), \sum_{e \in \pi' \setminus \pi} (q_e + q'_e) \text{wgt}(e) &\leq \sum_{e \in \pi' \setminus \pi} (q_e + q'_e + 1) \text{wgt}(e). \\ &\Leftrightarrow \\ \forall \pi' \in \text{Paths}(\mathcal{A}), \sum_{e \in \pi' \setminus \pi} q_e \text{wgt}(e) &\leq \sum_{e \in \pi' \setminus \pi} (q_e + 1) \text{wgt}(e), \end{aligned}$$

where the equivalence holds by Lemma 4. This already holds for $\pi \in \text{Paths}(\mathcal{A})$ such that $p_\pi > 0$. Thus any path π such that $p'_\pi > 0$ must satisfy $\pi \in S_{\vec{p}}$, which is equivalent to the above. So any period vector \vec{p}' for the base \vec{p} satisfies $\bigwedge_{\pi \notin S_{\vec{p}}} p'_\pi = 0$. Conversely, for any such vector \vec{p}' , $\vec{p} + k\vec{p}'$ is a Nash equilibrium for all $k \geq 0$. ◀

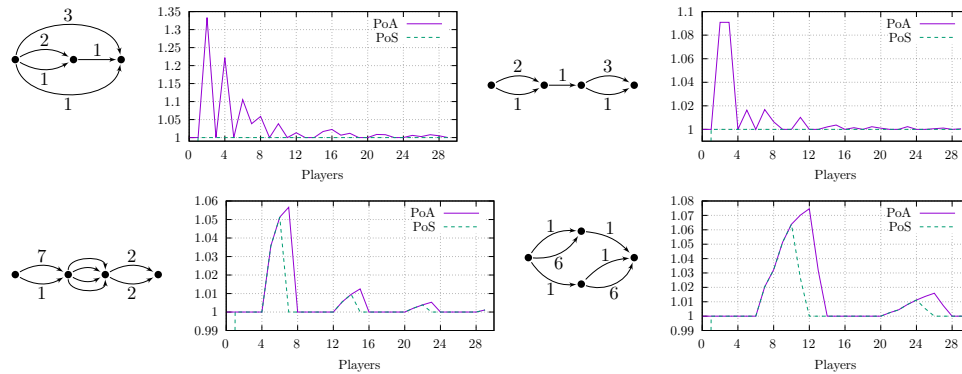
Given the pair \vec{p}, \vec{q} , we write another query to guess m, \vec{p}' such that $\text{NE}(\vec{p}') \wedge \text{flow}(\vec{p}') = m\vec{\delta} \wedge \bigwedge_{\pi \notin S_{\vec{p}}} p'_\pi = 0$. If this is satisfiable, then we query again the solver to find the smallest such m , and keep the set $L(\vec{q}, m\vec{\delta})$. Otherwise \vec{q} is added to B .

Price of Anarchy and Stability

Given a semilinear representation $B \cup \bigcup_{i \in I} L(\vec{b}^i, \vec{\delta})$ of $\text{LocOpt}(\mathcal{A})$, observe that there is only a finite number of vectors with a given n number of players. So in order to compute the *global* social optimum with n players, we iterate over all vectors with n players in this representation, and keep the minimal social cost. We first iterate over $\{\vec{b} \in B \mid \|\vec{b}\| = n\}$ and consider the vector with the least social cost among those (this set can be empty). Second, for each linear set $L(\vec{b}^i, \vec{\delta})$ such that $\frac{n - \|\vec{b}^i\|}{\|\vec{\delta}\|}$ is an integer, we compute the vector $\vec{b}^i + \frac{\|\vec{b}^i\| - n}{\|\vec{\delta}\|} \vec{\delta}$, compute its social cost, and keep it if it is less than the previous value.

We compute the social costs of the best and the worst Nash equilibria similarly on the semilinear representation of $\text{flow}(\text{NE}_{\mathcal{A}})$.

Figure 5 shows the plots of the PoA and PoS computed for four examples. We used the Python `sympy` package for solving linear equations, and the Z3 SMT solver for integer arithmetic queries. The characteristic vector was always easy to compute since it is computable in polynomial time in the size of the linear equation system. However, the number of base vectors can be large and depends on the weights used in the network. Our prototype is currently limited in scalability but it allows us to explore small yet non-trivial networks.



■ **Figure 5** Plots for PoA and PoS on four series-parallel networks. Unlabeled edges have weight 1.

6 More on Related Works

Inefficiency in congestion networks were mentioned in [30], and equilibria were first mathematically studied in [36]. Network congestion games are mainly studied in two settings which have different mathematical properties: the *nonatomic* case, where one considers a large number of players, each of which contributes an infinitesimal amount to congestion; and the *atomic* case, as we do, where there is a discrete number of players involved.

Existence and properties of Nash equilibria in the nonatomic case were established in [4]. The price of anarchy of the nonatomic case was studied in [32] which gives a tight bound of $\frac{4}{3}$ for networks with affine cost functions. [28, Chapter 18] presents a survey of these results. It is shown in [17] that Nash equilibria in atomic network congestion games can be found in polynomial time, by reduction to maximum flow in the symmetric case, that is, when all players share the same source and target vertices. The problem in the non-symmetric case is however complete for the class PLS, Polynomially Local Search, and is believed to be intractable [26, 29]. In extension-parallel networks, best-response procedures are shown to converge in linear time in [19]; but this does not extend to general series-parallel graphs.

The complexity of finding *extremal* Nash equilibria, that is, the best and the worst ones is however higher. Finding such equilibria is NP-hard for the makespan objective with varying sizes [20]. For the makespan objective and unit sizes, finding a Nash equilibrium minimizing the makespan in series-parallel networks with linear cost functions is strongly NP-hard when the number of players is part of the input, while a polynomial-time greedy algorithm allows one to find a worst Nash equilibrium [21]. For the total cost objective (as in this paper), NP-hardness holds for both best and worst equilibria for three and two players respectively [33]. Note that in our work, we are interested in computing such equilibria (or their costs) for arbitrarily large numbers of players. In the more general case of network congestion games with *dynamic* strategies which allow players to choose each move according to the current state of the game, doubly exponential-time algorithms were given for computing such equilibria in [5] when the number of players is encoded in binary.

It is possible to efficiently compute the social optimum in atomic congestion games by transforming the cost function, and reducing the problem to the computation of Nash equilibria [15]. In our case, this transformation would yield affine cost functions. This direction could be exploited to compute the costs of socially optimal profiles using semilinear representations for Nash equilibria, if these could be extended to affine costs. The behaviors of PoA for large numbers of players have been studied before. [18] considers congestion games with large numbers of players, and shows that the PoA of atomic congestion games converges to the PoA of the nonatomic game; the result holds for games with affine cost functions and positive coefficients (as in our case). A consequence is that, although the PoA for the atomic case is often larger than that in the nonatomic case, this difference vanishes in the limit, and thus the upper bound of $\frac{4}{3}$ holds for the atomic case in the limit. In [13], the limit of atomic congestion games is considered in a setting where either players participate in the game with given probabilities that tend to 0, or they have weights that tend to 0; in both cases, the limit of the PoA for mixed equilibria is equal to that in a corresponding nonatomic game. Asymptotic PoA bounds (of ≈ 1.35188) are provided in [14] for symmetric atomic congestion games with affine cost functions, by restricting to specific strategies called k -uniform. In the nonatomic case, the works [11, 10] establish that the limit of the PoA is 1. Paper [12] considers nonatomic congestion games as a function of the demand and studies continuous derivability properties of the PoA function.

7 Conclusion

Our results provide theoretical tools that allow us to have a better understanding of the structure of Nash equilibria and social optima in atomic congestion games over series-parallel networks. An immediate question is how the semilinear representations would change if we allow affine cost functions rather than linear ones. However, extending further our approach to nonlinear cost functions would not be immediate since these sets would no longer be definable in Presburger arithmetic.

Although the characteristic vector is easy to compute, the exact computation of the whole semilinear representations is costly and currently does not scale to large networks. One might investigate how this computation can be rendered more efficient in practice. Another possible direction is to explore more efficient approximation algorithms using just the characteristic vector, and perhaps a subset of the base vectors.

References

- 1 Eitan Altman, Anurag Kumar, and Yezekael Hayel. A potential game approach for uplink resource allocation in a multichannel wireless access network. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 1–9, 2009.
- 2 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 295–304, 2004. doi:10.1109/FOCS.2004.68.
- 3 Baruch Awerbuch, Yossi Azar, and Amir Epstein. The price of routing unsplittable flow. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, STOC'05*, pages 57–66, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1060590.1060599.
- 4 Martin J. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. RAND Corporation, Santa Monica, CA, 1955.
- 5 Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur. Dynamic network congestion games. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, Goa, India, December 2020. URL: <https://hal.archives-ouvertes.fr/hal-02980833>.
- 6 I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976. URL: <http://www.jstor.org/stable/2041711>.
- 7 Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, November 2011. doi:10.1007/s00453-010-9427-8.
- 8 George Christodoulou and Elias Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games,. In *Proceedings of the 13th Annual European Conference on Algorithms, ESA'05*, pages 59–70, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11561071_8.
- 9 George Christodoulou and Elias Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, STOC'05*, pages 67–73, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1060590.1060600.
- 10 Riccardo Colini-Baldeschi, Roberto Cominetti, Panayotis Mertikopoulos, and Marco Scarsini. When is selfish routing bad? the price of anarchy in light and heavy traffic. *Oper. Res.*, 68(2):411–434, 2020. doi:10.1287/opre.2019.1894.
- 11 Riccardo Colini-Baldeschi, Roberto Cominetti, and Marco Scarsini. Price of anarchy for highly congested routing games in parallel networks. *Theory Comput. Syst.*, 63(1):90–113, 2019. doi:10.1007/s00224-017-9834-1.
- 12 Roberto Cominetti, Valerio Dose, and Marco Scarsini. The price of anarchy in routing games as a function of the demand. *Mathematical Programming*, 2021. To appear. doi:10.1007/s10107-021-01701-7.
- 13 Roberto Cominetti, Marco Scarsini, Marc Schröder, and NE Stier-Moses. Convergence of large atomic congestion games. *arXiv preprint*, 2020. arXiv:2001.02797.
- 14 Jasper de Jong, Walter Kern, Berend Steenhuisen, and Marc Uetz. The asymptotic price of anarchy for k-uniform congestion games. In *International Workshop on Approximation and Online Algorithms*, pages 317–328. Springer, 2017.
- 15 Alberto Del Pia, Michael Ferris, and Carla Michini. Totally unimodular congestion games. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 577–588. SIAM, 2017.
- 16 Samuel Eilenberg and Marcel Paul Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13:173–191, 1969.

- 17 Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC '04*, pages 604–612, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007445.
- 18 Michal Feldman, Nicole Immorlica, Brendan Lucier, Tim Roughgarden, and Vasilis Syrgkanis. The price of anarchy in large games. In *48th ACM Symposium on Theory of Computing (STOC) 2016*, June 2016. URL: <https://www.microsoft.com/en-us/research/publication/price-anarchy-large-games/>.
- 19 Dimitris Fotakis. Congestion games with linearly independent paths: Convergence time and price of anarchy. In *Proceedings of the 1st International Symposium on Algorithmic Game Theory, SAGT '08*, pages 33–45, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-79309-0_5.
- 20 Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul Spirakis. The structure and complexity of nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009. Graphs, Games and Computation: Dedicated to Professor Burkhard Monien on the Occasion of his 65th Birthday. doi:10.1016/j.tcs.2008.01.004.
- 21 Elisabeth Gassner, Johannes Hatzl, Sven O. Krumke, Heike Sperber, and Gerhard J. Woeginger. How hard is it to find extreme nash equilibria in network congestion games? *Theoretical Computer Science*, 410(47):4989–4999, 2009. doi:10.1016/j.tcs.2009.07.046.
- 22 Seymour Ginsburg and Edwin H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964. URL: <http://www.jstor.org/stable/1994067>.
- 23 Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, July 2018. doi:10.1145/3242953.3242964.
- 24 Bainian Halo and Carla Michini. The price of anarchy in series-parallel network congestion games. Technical report, University of Wisconsin-Madison, 2021.
- 25 Ryuichi Ito. Every semilinear set is a finite union of disjoint linear sets. *J. Comput. Syst. Sci.*, 3(2):221–231, May 1969. doi:10.1016/S0022-0000(69)80014-0.
- 26 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.
- 27 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009. doi:10.1016/j.cosrev.2009.04.003.
- 28 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.
- 29 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.
- 30 A. C. Pigou. *Economics of welfare*. McMillan, 1920.
- 31 Robert W. Rosenthal. The network equilibrium problem in integers. *Networks*, 3:53–59, 1973.
- 32 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.
- 33 Heike Sperber. How to find nash equilibria with extreme total latency in network congestion games? In *2009 International Conference on Game Theory for Networks*, pages 158–163, 2009. doi:10.1109/GAMENETS.2009.5137397.
- 34 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, STOC '79*, pages 1–12, New York, NY, USA, 1979. Association for Computing Machinery. doi:10.1145/800135.804393.
- 35 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978.

- 36 John Glen Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the institution of civil engineers*, 1(3):325–362, 1952.
- 37 Zijun Wu, Rolf H Moehring, Chunying Ren, and Dachuan Xu. A convergence analysis of the price of anarchy in atomic congestion games. *Mathematical Programming*, 2022. To appear. doi:10.1007/s10107-022-01853-0.

A Proofs of Section 3

A.1 Proofs of Section 3.1 and 3.2

► **Lemma 1.** *In a network congestion game $\langle \mathcal{A}, n \rangle$, a flow \vec{q} is locally-optimal if, and only if, for all $\pi, \pi' \in \text{Paths}_{\mathcal{A}}$ such that $\forall e \in \pi, q_e > 0$,*

$$\sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot (2q_e - 1) \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (2q_e + 1). \quad (3)$$

Proof. Observe that flow \vec{q} is locally optimal iff for all paths π, π' such that $\forall e \in \pi, q_e > 0$, the vector \vec{q}' defined by

$$q'_e = \begin{cases} q_e - 1 & \text{if } e \in \pi \setminus \pi', \\ q_e + 1 & \text{if } e \in \pi' \setminus \pi, \\ q_e & \text{otherwise,} \end{cases}$$

satisfies $\text{soccost}(\vec{q}) \leq \text{soccost}(\vec{q}')$. Given such paths π, π' , let us thus write this inequality as

$$\begin{aligned} \sum_{e \in E} \text{wgt}(e) \cdot q_e^2 &\leq \sum_{e \in E} \text{wgt}(e) \cdot q_e'^2 \\ &= \sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot (q_e - 1)^2 + \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (q_e + 1)^2 + \sum_{e \in \pi \cap \pi'} \text{wgt}(e) \cdot q_e^2. \end{aligned}$$

This is equivalent to

$$\sum_{e \in \pi \setminus \pi' \cup \pi' \setminus \pi} \text{wgt}(e) \cdot q_e^2 \leq \sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot (q_e^2 - 2q_e + 1) + \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (q_e^2 + 2q_e + 1),$$

hence to

$$\sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot (2q_e - 1) \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (2q_e + 1). \quad \blacktriangleleft$$

► **Lemma 4.** *In a series-parallel network \mathcal{A} , for all period vectors $\vec{d} \in \mathbb{N}^E$ of a semilinear representation of $\text{LocOpt}_{\geq n_0}(\mathcal{A})$, there exists $\kappa \geq 0$ such that for all $\pi \in \text{Paths}_{\mathcal{A}}$, we have $\sum_{e \in \pi} \text{wgt}(e) \cdot d_e = \kappa$.*

Proof. Consider a linear set $L(\vec{b}, \vec{d})$ in $\text{LocOpt}_{\geq n_0}(\mathcal{A})$ and two paths π_1 and π_2 .

Applying Lemma 3 to $\vec{b} \in \text{LocOpt}_{\geq n_0}(\mathcal{A})$, we have that $b_e > 0$ for all $e \in E$. For any $k \geq 0$, the flow $\vec{q} = \vec{b} + k\vec{d}$ is locally optimal and has $q_e > 0$ for all $e \in \pi_2$. By Lemma 1,

$$\sum_{e \in \pi_2 \setminus \pi_1} \text{wgt}(e) \cdot (2(b_e + kd_e) - 1) \leq \sum_{e \in \pi_1 \setminus \pi_2} \text{wgt}(e) \cdot (2(b_e + kd_e) + 1)$$

which rewrites as

$$\sum_{e \in \pi_2 \setminus \pi_1} \text{wgt}(e) \cdot (2b_e - 1) - \sum_{e \in \pi_1 \setminus \pi_2} \text{wgt}(e) \cdot 2(b_e + 1) + 2k \left(\sum_{e \in \pi_2} \text{wgt}(e) d_e - \sum_{e \in \pi_1} \text{wgt}(e) d_e \right) \leq 0.$$

Since this holds for any $k \geq 0$, we must have $\sum_{e \in \pi_2} \text{wgt}(e)d_e - \sum_{e \in \pi_1} \text{wgt}(e)d_e \leq 0$. The converse inequality can be obtained using similar arguments, hence $\sum_{e \in \pi_2} \text{wgt}(e)d_e = \sum_{e \in \pi_1} \text{wgt}(e)d_e$. \blacktriangleleft

B Proofs of Section 4

► **Lemma 9.** *Given a network \mathcal{A} , a strategy profile \vec{p} is a Nash equilibrium if, and only if,*

$$\forall \pi, \pi' \in \text{Paths}, p_\pi > 0 \implies \sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot q_e \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (q_e + 1), \quad (6)$$

where $\vec{q} = \text{flow}(\vec{p})$.

Proof. The lemma is a reformulation of (2): for $\pi, \pi' \in \text{Paths}$ with $p_\pi > 0$, $\text{cost}_\pi(\vec{p}) \leq \text{cost}_{\pi'}(\vec{p})$ can be written as

$$\sum_{e \in \pi} \text{wgt}(e) \cdot q_e \leq \sum_{e \in \pi'} \text{wgt}(e) \cdot q'_e,$$

where \vec{q} and \vec{q}' are the respective flows of \vec{p} and \vec{p}' . This is equivalent to

$$\sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot q_e \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot q'_e = \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (q_e + 1). \quad \blacktriangleleft$$

► **Lemma 10.** *The sets $\text{NE}(\mathcal{A})$ and $\text{flow}(\text{NE}(\mathcal{A}))$ are semilinear.*

Proof. We show that both sets can be expressed in Presburger arithmetic. This follows from Lemma 9 since the following formula with free variables $\{q_e \mid e \in E\} \cup \{p_\pi \mid \pi \in \text{Paths}(\mathcal{A})\}$ expresses (6) in Presburger arithmetic:

$$\phi = \bigwedge_{\pi, \pi' \in \text{Paths}} \left(p_\pi > 0 \implies \sum_{e \in \pi \setminus \pi'} \text{wgt}(e) \cdot q_e \leq \sum_{e \in \pi' \setminus \pi} \text{wgt}(e) \cdot (q_e + 1) \right) \wedge \bigwedge_{e \in E} \left(q_e = \sum_{\pi \in \text{Paths}: e \in \pi} p_\pi \right).$$

Here we use the fact that Paths is finite so that the above is a well-defined formula. Now, existentially quantifying $\{q_e\}_{e \in E}$ in ϕ yields a formula describing $\text{NE}(\mathcal{A})$. Existentially quantifying $\{p_\pi\}_{\pi \in \text{Paths}(\mathcal{A})}$ in ϕ yields $\text{flow}(\text{NE}(\mathcal{A}))$. \blacktriangleleft

Our next results prove that the “projections” of the Nash equilibria of series-parallel networks onto their constituent subnets still are Nash equilibria. We first formally define those projections.

For a network $\mathcal{A} = \mathcal{A}_1; \mathcal{A}_2$ and $\vec{p} \in \mathfrak{S}(\mathcal{A})$, for $i \in \{1, 2\}$, let us define $\text{prj}_{\mathcal{A}_i}(\vec{p}) \in \mathfrak{S}(\mathcal{A}_i)$ where for each $\pi_i \in \text{Paths}_{\mathcal{A}_i}$, $\text{prj}_{\mathcal{A}_i}(\vec{p})_{\pi_i} = \sum_{\pi_{3-i} \in \text{Paths}_{\mathcal{A}_{3-i}}} p_{\pi_1 \pi_2}$. Thus, $\text{prj}_{\mathcal{A}_i}(\vec{p})_{\pi_i}$ is the number of players that cross the path π_i in the profile \vec{p} .

For a network $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$, and a vector $\vec{p} \in \mathfrak{S}(\mathcal{A})$, for $i \in \{1, 2\}$, let us denote $\vec{p}|_{\mathcal{A}_i} \in \mathfrak{S}(\mathcal{A}_i)$ obtained by restricting \vec{p} to $\text{Paths}_{\mathcal{A}_i}$. Similarly, for a vector $\vec{q} \in \mathcal{F}(\mathcal{A})$, let $\vec{q}|_{\mathcal{A}_i}$ the restriction of \vec{q} to the edges of \mathcal{A}_i .

► **Lemma 16.** *Consider a network $\mathcal{A} = \mathcal{A}_1; \mathcal{A}_2$. Then, for all $\vec{p} \in \mathfrak{S}(\mathcal{A})$, we have*

$$\vec{p} \in \text{NE}(\mathcal{A}) \Leftrightarrow \forall i \in \{1, 2\}, \text{prj}_{\mathcal{A}_i}(\vec{p}) \in \text{NE}(\mathcal{A}_i).$$

32:20 Semilinear Representations for Series-Parallel Atomic Congestion Games

Proof. Consider $\vec{p} \in \mathfrak{S}(\mathcal{A})$. Observe that all $\pi \in \text{Paths}_{\mathcal{A}}$ can be written as $\pi = \pi_1\pi_2$ where $\pi_i \in \text{Paths}_{\mathcal{A}_i}$, and that $\text{cost}_{\pi}(\vec{p}) = \text{cost}_{\pi_1}(\text{prj}_{\mathcal{A}_1}(\vec{p})) + \text{cost}_{\pi_2}(\text{prj}_{\mathcal{A}_2}(\vec{p}))$.

Assume that $\vec{p} \in \text{NE}(\mathcal{A})$. Consider $\pi_1 \in \text{Paths}_{\mathcal{A}_1}$ such that $\text{prj}_{\mathcal{A}_1}(\vec{p})_{\pi_1} > 0$. Then, there must exist a path $\pi_2 \in \text{Paths}_{\mathcal{A}_2}$ such that $p_{\pi_1\pi_2} > 0$. By (2), we have that for all $\pi' \in \text{Paths}_{\mathcal{A}}$, we have $\text{cost}_{\pi_1\pi_2}(\vec{p}) \leq \text{cost}_{\pi'}(\vec{p}')$ where $\vec{p}' = \vec{p} - \pi_1\pi_2 + \pi'$. In particular, for all $\pi'_1 \in \text{Paths}_{\mathcal{A}_1}$, we have $\text{cost}_{\pi_1\pi_2}(\vec{p}) \leq \text{cost}_{\pi'_1\pi_2}(\vec{p}')$, i.e.,

$$\text{cost}_{\pi_1}(\text{prj}_{\mathcal{A}_1}(\vec{p})) + \text{cost}_{\pi_2}(\text{prj}_{\mathcal{A}_2}(\vec{p})) \leq \text{cost}_{\pi'_1}(\text{prj}_{\mathcal{A}_1}(\vec{p}')) + \text{cost}_{\pi_2}(\text{prj}_{\mathcal{A}_2}(\vec{p}')).$$

The second terms of both sides are equal since $\text{prj}_{\mathcal{A}_2}(\vec{p}) = \text{prj}_{\mathcal{A}_2}(\vec{p}')$. It follows that

$$\text{cost}_{\pi_1}(\text{prj}_{\mathcal{A}_1}(\vec{p})) \leq \text{cost}_{\pi'_1}(\text{prj}_{\mathcal{A}_1}(\vec{p}')) = \text{cost}_{\pi'_1}(\text{prj}_{\mathcal{A}_1}(\vec{p}) - \pi_1 + \pi'_1),$$

which means that $\text{prj}_{\mathcal{A}_1}(\vec{p}) \in \text{NE}(\mathcal{A}_1)$. The argument is symmetric for \mathcal{A}_2 .

Conversely, assume that for all $i \in \{1, 2\}$, $\text{prj}_{\mathcal{A}_i}(\vec{p}) \in \text{NE}(\mathcal{A}_i)$. Consider any path $\pi \in \text{Paths}_{\mathcal{A}}$ such that $p_{\pi} > 0$, and write it as $\pi = \pi_1\pi_2$. Then, for both $i \in \{1, 2\}$, $\text{prj}_{\mathcal{A}_i}(\vec{p})_{\pi_i} > 0$. Take any other path $\pi' = \pi'_1\pi'_2 \in \text{Paths}_{\mathcal{A}}$. Because each $\text{prj}_{\mathcal{A}_i}(\vec{p})$ is a Nash equilibrium, we have

$$\text{cost}_{\pi_i}(\text{prj}_{\mathcal{A}_i}(\vec{p})) \leq \text{cost}_{\pi'_i}(\text{prj}_{\mathcal{A}_i}(\vec{p}) - \pi_i + \pi'_i).$$

Summing up these inequations, we get

$$\begin{aligned} \text{cost}_{\pi_1\pi_2}(\vec{p}) &\leq \text{cost}_{\pi'_1}(\text{prj}_{\mathcal{A}_1}(\vec{p}) - \pi_1 + \pi'_1) + \text{cost}_{\pi'_2}(\text{prj}_{\mathcal{A}_2}(\vec{p}) - \pi_2 + \pi'_2) \\ &= \text{cost}_{\pi'_1}(\text{prj}_{\mathcal{A}_1}(\vec{p} - \pi_1\pi_2 + \pi'_1\pi'_2)) + \text{cost}_{\pi'_2}(\text{prj}_{\mathcal{A}_2}(\vec{p} - \pi_1\pi_2 + \pi'_1\pi'_2)) \\ &= \text{cost}_{\pi'}(\vec{p} - \pi + \pi'). \end{aligned}$$

Hence π' is not a profitable deviation, whichever $\pi' \in \text{Paths}_{\mathcal{A}}$. ◀

► **Lemma 17.** Consider a network $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$. Then, for all $\vec{p} \in \text{NE}(\mathcal{A})$, we have that for all $i \in \{1, 2\}$, we have $\vec{p}|_{\mathcal{A}_i} \in \text{NE}(\mathcal{A}_i)$.

Proof. Consider $\vec{p} \in \text{NE}(\mathcal{A})$, and $i \in \{1, 2\}$. Then, for all $\pi, \pi' \in \text{Paths}_{\mathcal{A}_i}$ such that $p_{\pi} > 0$, $\text{cost}_{\pi}(\vec{p}) \leq \text{cost}_{\pi'}(\vec{p} - \pi + \pi')$. Since the only paths sharing edges with π and π' are in $\text{Paths}_{\mathcal{A}_i}$, we have that $\text{cost}_{\pi}(\vec{p}|_{\mathcal{A}_i}) \leq \text{cost}_{\pi'}(\vec{p}|_{\mathcal{A}_i} - \pi + \pi')$. Hence $\vec{p}|_{\mathcal{A}_i} \in \text{NE}(\mathcal{A}_i)$. ◀

► **Remark 18.** Notice that contrary to Lemma 16, Lemma 17 is not an equivalence: a 2-player strategy profile involving the “shortest” path of \mathcal{A}_1 with the “shortest” path of \mathcal{A}_2 need not yield a Nash equilibrium.

Playing (Almost-)Optimally in Concurrent Büchi and Co-Büchi Games

Benjamin Bordais

Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Patricia Bouyer

Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Stéphane Le Roux

Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Abstract

We study two-player concurrent stochastic games on finite graphs, with Büchi and co-Büchi objectives. The goal of the first player is to maximize the probability of satisfying the given objective. Following Martin’s determinacy theorem for Blackwell games, we know that such games have a value. Natural questions are then: does there exist an optimal strategy, that is, a strategy achieving the value of the game? what is the memory required for playing (almost-)optimally?

The situation is rather simple to describe for turn-based games, where positional pure strategies suffice to play optimally in games with parity objectives. Concurrency makes the situation intricate and heterogeneous. For most ω -regular objectives, there do indeed not exist optimal strategies in general. For some objectives (that we will mention), infinite memory might also be required for playing optimally or almost-optimally.

We also provide characterizations of local interactions of the players to ensure positionality of (almost-)optimal strategies for Büchi and co-Büchi objectives. This characterization relies on properties of game forms underpinning the formalism for defining local interactions of the two players. These well-behaved game forms are like elementary bricks which, when they behave well in isolation, can be assembled in graph games and ensure the good property for the whole game.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Concurrent Games, Optimal Strategies, Büchi Objective, co-Büchi Objective

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.33

Related Version *Extended Version*: <https://arxiv.org/pdf/2203.06966.pdf> [3]

1 Introduction

Stochastic concurrent games. Games on graphs are an intensively studied mathematical tool, with wide applicability in verification and in particular for the controller synthesis problem, see for instance [16, 1]. We consider two-player stochastic concurrent games played on finite graphs. For simplicity (but this is with no restriction), such a game is played over a finite bipartite graph called an arena: some states belong to Nature while others belong to the players. Nature is stochastic, and therefore assigns a probabilistic distribution over the players’ states. In each players’ state, a local interaction between the two players (called Player A and Player B) happens, specified by a two-dimensional table. Such an interaction is resolved as follows: Player A selects a probability distribution over the rows while Player B selects a probability distribution over the columns of the table; this results into a distribution over the cells of the table, each one pointing to a Nature state of the graph. An example of game arena (with no Nature states – we could add dummy deterministic Nature states) is given in Figure 1 (this example comes from [10]). At state q_0 , the interaction between the two players is given by the table, and each player has two actions: if Player A plays the second row and Player B the first column, then the game proceeds to state \top : in that case, the game always goes back to q_0 .



© Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 33; pp. 33:1–33:18



Leibniz International Proceedings in Informatics

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

Globally, the game proceeds as follows: starting at an initial state q_0 , the two players play in the local interaction of the current state, and the joint choice determines (stochastically) the next Nature state of the game, itself moving randomly to players' states; the game then proceeds subsequently from the new players' state. The way players make choices is given by strategies, which, given the sequence of states visited so far (the so-called history), assign local strategies for the local interaction of the state the game is in. For application in controller synthesis, strategies will correspond to controllers, hence it is desirable to have strategies simple to implement. We will be in particular interested in strategies which are *positional*, i.e. strategies which only depend on the current state of the game, not on the whole history. When each player has fixed a strategy (say s_A for Player A and s_B for Player B), this defines a probability distribution $\mathbb{P}_{s_A, s_B}^{q_0}$ over infinite sequences of states of the game. The objectives of the two players are opposite (we assume a zero-sum setting): together with the game, a measurable set W of infinite sequences of states is fixed; the objective of Player A is to maximize the probability of W while the objective of Player B is to minimize it.

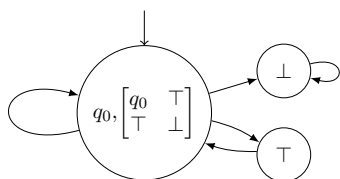
Back to the example of Figure 1. If Player A (resp. B) plays the first row (resp. column) with probability p_A (resp. p_B), then the probability to move to \perp in one step is $(1-p_A) \cdot (1-p_B)$. If Player A repeatedly plays the same strategy at q_0 with $p_A < 1$, by playing $p_B = 0$, Player B will enforce \perp almost-surely; however, if Player A plays $p_A = 1$, then by playing $p_B = 1$, Player B enforces staying in q_0 , hence visiting \top with probability 0. On the contrary Player A can ensure visiting \top infinitely often with probability $1 - \varepsilon$ for every $\varepsilon > 0$, by playing iteratively at q_0 the first row of the table with probability $1 - \varepsilon_k$ (k the number of visits to q_0) and the second row with probability ε_k , where the sequence $(\varepsilon_k)_k$ decreases fast to zero (see Appendix A).

Values and (almost-)optimal strategies. As mentioned above, Player A wants to maximize the probability of W , while Player B wants to minimize this probability. Formally, given a strategy s_A for Player A, its value is measured by $\inf_{s_B} \mathbb{P}_{s_A, s_B}^{q_0}(W)$, and Player A wants to maximize that value. Dually, given a strategy s_B for Player B, its value is measured by $\sup_{s_A} \mathbb{P}_{s_A, s_B}^{q_0}(W)$, and Player B wants to minimize that value. Following Martin's determinacy theorem for Blackwell games [14], it actually holds that the game has a *value* given by

$$\chi_{q_0} = \sup_{s_A} \inf_{s_B} \mathbb{P}_{s_A, s_B}^{q_0}(W) = \inf_{s_B} \sup_{s_A} \mathbb{P}_{s_A, s_B}^{q_0}(W)$$

While this ensures the existence of almost-optimal strategies (that is, ε -optimal strategies for every $\varepsilon > 0$) for both players, it says nothing about the existence of optimal strategies, which are strategies achieving χ_{q_0} . In general, except for safety objectives, optimal strategies may not exist, as witnessed by the example of Figure 1, which uses a Büchi condition. Also, it says nothing about the complexity of optimal strategies (when they exist) and ε -optimal strategies. Complexity of a strategy is measured in terms of memory that is used by the strategy: while general strategies may depend on the whole history of the game, a *positional* strategy only depends on the current state of the game; a *finite-memory* strategy records a finite amount of information using a finite automaton; the most complex ones, the *infinite-memory* strategies require more than a finite automaton to record information necessary to take decisions.

Back to the game of Figure 1, assuming the Büchi condition “visit \top infinitely often”, the game is such that $\chi_{q_0} = 1$. However Player A has no optimal strategy, and can only achieve $1 - \varepsilon$ for every $\varepsilon > 0$ with an infinite-memory strategy, and any positional strategy has value 0.



■ **Figure 1** A concurrent game with objective Büchi($\{T\}$).

$$\mathcal{F}_{q_0} = \begin{bmatrix} x & y \\ y & z \end{bmatrix}$$

■ **Figure 2** The local interaction at state q_0 .

$$\langle \mathcal{F}_{q_0}, \mu_v \rangle = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

■ **Figure 3** The local interaction at state q_0 valued by the vector μ_v giving the value of states.

The contributions of this work. We are interested in memory requirements for optimal and ε -optimal strategies in concurrent games with parity objectives. The situation is rather heterogeneous: while safety objectives enjoy very robust properties (existence of positional optimal strategies in all cases, see for instance [11, Thm. 1]), the situation appears as much more complex for parity objectives (already with three colors): there may not exist optimal strategies, and when they exist, optimal strategies as well as ε -optimal strategies require in general infinite memory (the case of optimal strategies was proven in [10] while the case of ε -optimal strategies is a consequence of the Büchi case, studied in [11, Thm. 2]). The case of reachability objectives was studied with details in [2]: optimal strategies may not exist, but there exists a positional strategy that is 1) optimal from each state from where there exists an optimal strategy, and 2) ε -optimal from the other states.

In this paper, we focus on Büchi and co-Büchi objectives. Few things were known for those games: specifically, it was shown in [11, Thm. 2] that Büchi and co-Büchi concurrent games may have no optimal strategies, and that ε -optimal strategies may require infinite memory for Büchi objectives. We show in addition that, when optimal strategies exist everywhere, optimal strategies can be chosen positional for Büchi objectives but may require infinite memory for co-Büchi objectives. We more importantly characterize “well-behaved” local interactions (i.e. interactions of the two players at each state, which are given by tables) for ensuring positionality of (ε -)optimal strategies in the various settings where they do not exist in the general case. We follow the approach used in [2] for reachability objectives and abstract those local interactions into game forms, where cells of the table are now seen as variables (some of them being equal). For instance, the game form associated with state q_0 in the running example has three outcomes: x , y and z , and it is given in Figure 2. Game forms can be seen as elementary bricks that can be used to build games on graphs. Given a property we want to hold on concurrent games (e.g. the existence of positional optimal strategies in Büchi games), we characterize those bricks that are safe w.r.t. that property, that is the game forms that behave well when used individually, the ones ensuring that, when they are the only non-trivial local interaction in a concurrent game, the property holds. Then, we realize that they also behave well when used collectively: if all local interactions are safe in a concurrent game, then the whole game ensures the property of interest. We obtain a clear-cut separation: if all local interactions are safe in a concurrent game, then the property is necessarily ensured; on the other hand, if a game form is not safe, one can build a game where it is the only non-trivial local interaction that does ensure the property. Our contributions can be summarized as follows:

1. In the general setting of prefix-independent objectives, we characterize positional uniformly optimal strategies using locally optimal strategies (i.e. strategies which are optimal at local interactions) and constraints on values of end-components generated by the strategies (Lemma 16).

2. We study Büchi concurrent games. We first show that there is a positional uniformly optimal strategy in a Büchi game as soon as it is known that optimal strategies exist from every state (Proposition 17). To benefit from this result, we give a (sufficient and necessary) condition on (game forms describing) local interactions to ensure the existence of optimal strategies. In particular, if all local interactions are well-behaved (w.r.t. the condition), then it will be the case that positional uniformly optimal strategies exist (Theorem 19). We also give a weaker necessary and sufficient condition on local interactions to ensure the existence of positional ε -optimal strategies in Büchi games, since in general, infinite memory might be required (Theorem 22).
3. We study co-Büchi concurrent games. We first show that optimal strategies might require infinite memory in co-Büchi games, in contrast with Büchi games (Subsection 6.1). We also characterize local interactions that ensure the existence of positional optimal strategies in co-Büchi games (Theorem 25). It is not useful to do the same for positional ε -optimal strategies since it is always the case that such strategies exist [5].

Additional details and proofs can be found in the arXiv version of this paper [3].

All these results show the contrast between concurrent games and turn-based games: indeed, in the latter (which have attracted more attention these last years), pure (that is, deterministic) positional optimal strategies always exist for parity objectives (hence in particular, for Büchi and co-Büchi objectives) [15, 7, 18]. The results presented in this paper hence show the complexity inherent to concurrent interactions in games. Those have nevertheless retained some attention over the last twenty years [10, 5, 9, 6, 12], and are relevant in applications [13].

2 Game Forms

A *discrete probability distribution* over a non-empty finite set Q is a function $\mu : Q \rightarrow [0, 1]$ such that $\sum_{x \in Q} \mu(x) = 1$. The *support* $\text{Supp}(\mu)$ of a probabilistic distribution $\mu : Q \rightarrow [0, 1]$ corresponds to the set of non-zeros of the distribution: $\text{Supp}(\mu) = \{q \in Q \mid \mu(q) > 0\}$. The set of all distributions over the set Q is denoted $\mathcal{D}(Q)$.

Informally, game forms are two-dimensional tables with variables while games in normal forms are game forms whose outcomes are real values in $[0, 1]$. Formally:

► **Definition 1** (Game form and game in normal form). *A game form (GF for short) is a tuple $\mathcal{F} = \langle \text{St}_A, \text{St}_B, \mathcal{O}, \varrho \rangle$ where St_A (resp. St_B) is a non-empty finite set of actions available to Player A (resp. B), \mathcal{O} is a non-empty set of outcomes, and $\varrho : \text{St}_A \times \text{St}_B \rightarrow \mathcal{O}$ is a function that associates an outcome to each pair of actions. When the set of outcomes \mathcal{O} is equal to $[0, 1]$, we say that \mathcal{F} is a game in normal form. For a valuation $v \in [0, 1]^{\mathcal{O}}$ of the outcomes, the notation $\langle \mathcal{F}, v \rangle$ refers to the game in normal form $\langle \text{St}_A, \text{St}_B, [0, 1], v \circ \varrho \rangle$.*

An example of game form (resp. game in normal form) is given in Figure 2 (resp. 3), where St_A (resp. St_B) are rows (resp. columns) of the table and x, y, z (resp. 0, 1) are the possible outcomes of the game form (resp. game in normal form). We use game forms to represent interactions between two players. The strategies available to Player A (resp. B) are convex combinations of actions given as the rows (resp. columns) of the table. In a game in normal form, Player A tries to maximize the outcome, whereas Player B tries to minimize it.

► **Definition 2** (Outcome of a game in normal form). *Let $\mathcal{F} = \langle \text{St}_A, \text{St}_B, [0, 1], \varrho \rangle$ be a game in normal form. The set $\mathcal{D}(\text{St}_A)$ (resp. $\mathcal{D}(\text{St}_B)$) is the set of (mixed) strategies available to Player A (resp. B). For a pair of strategies $(\sigma_A, \sigma_B) \in \mathcal{D}(\text{St}_A) \times \mathcal{D}(\text{St}_B)$, the outcome $\text{out}_{\mathcal{F}}(\sigma_A, \sigma_B)$ in \mathcal{F} of the strategies (σ_A, σ_B) is $\text{out}_{\mathcal{F}}(\sigma_A, \sigma_B) := \sum_{a \in \text{St}_A} \sum_{b \in \text{St}_B} \sigma_A(a) \cdot \sigma_B(b) \cdot \varrho(a, b) \in [0, 1]$.*

► **Definition 3** (Value of a game in normal form and optimal strategies). Let $\mathcal{F} = \langle \text{St}_A, \text{St}_B, [0, 1], \varrho \rangle$ be a game in normal form, and $\sigma_A \in \mathcal{D}(\text{St}_A)$ be a strategy for Player A. The value of strategy σ_A is $\text{val}_{\mathcal{F}}(\sigma_A) := \inf_{\sigma_B \in \mathcal{D}(\text{St}_B)} \text{out}_{\mathcal{F}}(\sigma_A, \sigma_B)$, and analogously for Player B, with a sup instead of an inf. When $\sup_{\sigma_A \in \mathcal{D}(\text{St}_A)} \text{val}_{\mathcal{F}}(\sigma_A) = \inf_{\sigma_B \in \mathcal{D}(\text{St}_B)} \text{val}_{\mathcal{F}}(\sigma_B)$, it defines the value of the game \mathcal{F} , denoted $\text{val}_{\mathcal{F}}$.

A strategy $\sigma_A \in \mathcal{D}(\text{St}_A)$ ensuring $\text{val}_{\mathcal{F}} = \text{val}_{\mathcal{F}}(\sigma_A)$ is said to be optimal. The set of all optimal strategies for Player A is denoted $\text{Opt}_A(\mathcal{F}) \subseteq \mathcal{D}(\text{St}_A)$, and analogously for Player B. Von Neumann's minimax theorem [17] ensures the existence of optimal strategies (for both players).

In the following, strategies in games in normal forms will be called GF-strategies, in order not to confuse them with strategies in concurrent (graph) games.

3 Concurrent Stochastic Games

We introduce the definition of a concurrent arena played on a finite graph, and of a concurrent game by adding a winning condition.

► **Definition 4** (Finite stochastic concurrent arena and game). A concurrent arena \mathcal{C} is a tuple $\langle Q, A, B, D, \delta, \text{dist} \rangle$ where Q is a non-empty set of states, A (resp. B) is the non-empty finite set of actions available to Player A (resp. B), D is the set of Nature states, $\delta : Q \times A \times B \rightarrow D$ is the transition function and $\text{dist} : D \rightarrow \mathcal{D}(Q)$ is the distribution function.

A concurrent game is a pair $\langle \mathcal{C}, W \rangle$ where \mathcal{C} is a concurrent arena and $W \subseteq Q^\omega$ is Borel. The set W is called the (winning) objective and it corresponds to the set of paths winning for Player A and losing for Player B.

For the rest of the section, we fix a concurrent game $\mathcal{G} = \langle \mathcal{C}, W \rangle$. An important class of objectives are the *prefix-independent* objectives, that is objectives W such that an infinite path is in W if and only if one of its suffixes is in W : for all $\rho \in Q^\omega$ and $\pi \in Q^+$, $\rho \in W \Leftrightarrow \pi \cdot \rho \in W$. In this paper, we more specifically focus on Büchi and co-Büchi objectives, which informally correspond to the infinite paths seeing infinitely often, or finitely often, a given set of states. We also recall the definitions of the reachability and safety objectives.

► **Definition 5** (Büchi and co-Büchi, Reachability and Safety objectives). Consider a target set of states $T \subseteq Q$. We define the following objectives:

- Büchi(T) := $\{\rho \in Q^\omega \mid \forall i \in \mathbb{N}, \exists j \geq i, \rho_j \in T\}$; Reach(T) := $\{\rho \in Q^\omega \mid \exists i \in \mathbb{N}, \rho_i \in T\}$;
- coBüchi(T) := $\{\rho \in Q^\omega \mid \exists i \in \mathbb{N}, \forall j \geq i, \rho_j \notin T\}$; Safe(T) := $\{\rho \in Q^\omega \mid \forall i \in \mathbb{N}, \rho_i \notin T\}$.

In concurrent games, game forms appear at each state and specify how the interaction of the Players determines the next (Nature) state. In fact, this corresponds to the local interactions of the game.

► **Definition 6** (Local interactions). The local interaction at state $q \in Q$ is the game form $\mathcal{F}_q = \langle A, B, D, \delta(q, \cdot, \cdot) \rangle$. That is, the GF-strategies available for Player A (resp. B) are the actions in A (resp. B) and the outcomes are the Nature states.

As an example, the local interaction at state q_0 in Figure 1 is represented in Figure 2, up to a renaming of the outcomes.

A strategy of a given Player then associates to every history (i.e. every finite sequence of states) a GF-strategy in the local interaction at the current state, in other words it associates a distribution on the actions available at the local interaction to the given Player.

► **Definition 7** (Strategies). A strategy for Player A is a function $s_A : Q^+ \rightarrow \mathcal{D}(A)$ such that, for all $\rho = q_0 \cdots q_n \in Q^+$, $s_A(\rho) \in \mathcal{D}(A)$ is a GF-strategy for Player A in the game form \mathcal{F}_{q_n} . A strategy $s_A : Q^+ \rightarrow \mathcal{D}(A)$ for Player A is positional if, for all $\pi = \rho \cdot q \in Q^+$ and $\pi' = \rho' \cdot q' \in Q^+$, if $q = q'$, then $s_A(\pi) = s_A(\pi')$ (that is, the strategy only depends on the current state of the game). We denote by S_C^A and PS_C^A the set of all strategies and positional strategies in arena \mathcal{C} for Player A. The definitions are analogous for Player B.

Before defining the outcome of the game given a strategy for a Player, we define the probability to go from state q to state q' , given two GF-strategies in the game form \mathcal{F}_q .

► **Definition 8** (Probability Transition). Let $q \in Q$ be a state and $(\sigma_A, \sigma_B) \in \mathcal{D}(A) \times \mathcal{D}(B)$ be two GF-strategies in the game form \mathcal{F}_q . For a state $q' \in Q$, the probability to go from q to q' if the players play σ_A and σ_B in q is equal to $\mathbb{P}_{\sigma_A, \sigma_B}(q, q') := \text{out}_{(\mathcal{F}_q, \text{dist}(\cdot)(q'))}(\sigma_A, \sigma_B)$.

From this, given two strategies, we deduce the probability of any cylinder supported by a finite path, and consequently of any Borel set in Q^ω . From a state $q_0 \in Q$, given two strategies s_A and s_B , this probability distribution is denoted $\mathbb{P}_{s_A, s_B}^{C, q_0} : \text{Borel}(Q) \rightarrow [0, 1]$. Let us now define the value of a strategy and of the game.

► **Definition 9** (Value of strategies and of the game). Let $s_A \in S_C^A$ be a Player A strategy. The value of s_A is the function $\chi_G[s_A] : Q \rightarrow [0, 1]$ such that for every $q \in Q$, $\chi_G[s_A](q) := \inf_{s_B \in S_C^B} \mathbb{P}_{s_A, s_B}^{C, q}[W]$.

The value for Player A is the function $\chi_G[A] : Q \rightarrow [0, 1]$ such that for all $q \in Q$, we have $\chi_G[A](q) := \sup_{s_A \in S_C^A} \chi_G[s_A](q)$. The value for Player B is defined similarly by reversing the supremum and infimum.

By Martin's result on the determinacy of Blackwell games [14], for all concurrent games $\mathcal{G} = \langle \mathcal{C}, W \rangle$, the values for both Players are equal, which defines the value of the game: $\chi_G := \chi_G[A] = \chi_G[B]$. Furthermore, a strategy $s_A \in S_C^A$ which ensures $\chi_G[s_A](q) = \chi_G(q)$ from some state $q \in Q$, is said to be optimal from q . If, in addition it ensures $\chi_G[s_A] = \chi_G$, it is said to be uniformly optimal.

We mention a very useful result of [4]. Informally, it shows that if there is a state with a positive value, then there is a state with value 1.

► **Theorem 10** (Theorem 1 in [4]). Let \mathcal{G} be a concurrent game with a prefix-independent objective. If there is a state $q \in Q$ such that $\chi_G(q) > 0$ (resp. $\chi_G(q) < 1$), then there is a state $q' \in Q$ such that $\chi_G(q') = 1$ (resp. $\chi_G(q') = 0$).

Finally, we define the Markov decision process which is induced by a positional strategy, and its end-components.

► **Definition 11** (Induced Markov decision process). Let $s_A \in PS_C^A$ be a positional strategy. The Markov decision process Γ (MDP for short) induced by the strategy s_A is the triplet $\Gamma := \langle Q, B, \iota \rangle$ where Q is the set of states, B is the set of actions and $\iota : Q \times B \rightarrow \mathcal{D}(Q)$ is a map associating to a state and an action a distribution over the states. For all $q \in Q$, $b \in B$ and $q' \in Q$, we set $\iota(q, b)(q') := \mathbb{P}_{q, q'}^{s_A(q), b}$.

The induced MDP Γ is a special case of a concurrent game where Player A does not play (A could be a singleton) and the set of Player B strategies is the same as in \mathcal{C} . The useful objects in MDPs are the end components [8], i.e. sub-MDPs that are strongly connected.

► **Definition 12** (End component and sub-game). Let $s_A \in PS_C^A$ be a positional strategy, and Γ its induced MDP. An end component (EC for short) H in Γ is a pair (Q_H, β) such that $Q_H \subseteq Q$ is a subset of states and $\beta : Q_H \rightarrow \mathcal{P}(B) \setminus \emptyset$ associates to each state a non-empty set of actions compatible with the EC H such that:

- for all $q \in Q_H$ and $b \in \beta(q)$, $\text{Supp}(\iota(q, b)) \subseteq Q_H$;
- the underlying graph (Q_H, E) is strongly connected, where $(q, q') \in E$ if and only if there is $b \in \beta(q)$ such that $q' \in \text{Supp}(\iota(q, b))$.

We denote by $D_H \subseteq D$ the set of Nature states compatible with the EC H : $D_H = \{d \in D \mid \text{Supp}(d) \subseteq Q_H\}$. Note that, for all $q \in Q_H$ and $b \in \beta(q)$, we have $\delta(q, \text{Supp}(s_A(q)), b) \subseteq D_H$.

The end component H can be seen as a concurrent arena. In that case, it is denoted $C_H^{s_A}$.

4 Uniform Optimality with Positional Strategies

In this section, we present a necessary and sufficient condition for a positional strategy to be uniformly optimal. Note that this result holds for any prefix-independent objective for which, in any finite MDP with the complement objective $Q^\omega \setminus W$, there is a positional optimal strategy. This is in particular the case for the Büchi and co-Büchi objectives (in fact, it holds for all parity objectives).

We assume a concurrent game $\mathcal{G} = \langle \mathcal{C}, W \rangle$ is given for this section, that W is Borel and prefix-independent, and that $v := \chi_{\mathcal{G}} \in [0, 1]^Q$ is the value (function) of the game. We first define the crucial notion of *local optimality*: informally, a Player A positional strategy is locally optimal if, at each local interaction, it is optimal in the game in normal form induced by the values of the game. As the outcomes of a local interaction are the Nature states, we have to be able to lift the valuation v of the states into a valuation of the Nature states. This is done via a convex combination in the definition below.

► **Definition 13** (Lifting a valuation of the states). *Let $w : Q \rightarrow [0, 1]$ be an arbitrary valuation of the states. We define $\mu_w : D \rightarrow [0, 1]$, the lift of the valuation w to Nature states in the following way: $\mu_w(d) := \sum_{q \in Q} \text{dist}(d)(q) \cdot w(q)$ for all $d \in D$.*

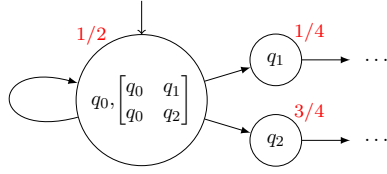
We can now define the local optimality of a positional strategy.

► **Definition 14** (Local optimality). *A Player A positional strategy $s_A \in \text{PS}_C^A$ is locally optimal if for all $q \in Q$: $\text{val}_{\langle \mathcal{F}_q, \mu_v \rangle}(s_A(q)) = v(q)$ (i.e. the GF-strategy $s_A(q)$ is optimal in $\langle \mathcal{F}_q, \mu_v \rangle$).*

Interestingly, in the MDP induced by a locally optimal strategy, all the states in a given EC have the same value (w.r.t. the valuation v). This is stated in the proposition below.

► **Proposition 15** (Proposition 18 in [2]). *For every locally optimal Player A positional strategy $s_A \in \text{PS}_C^A$, for all EC $H = (Q_H, \beta)$ in the MDP induced by the strategy s_A , there exists a value $v_H \in [0, 1]$ such that, for all $q \in Q_H$, we have $v(q) = v_H$.*

We now discuss how local optimality relates to uniform optimality. We first observe that local optimality is necessary for uniform optimality, and we illustrate this on the example of Figure 4: in this (partly depicted) game, Nature states are omitted, and values w.r.t. the valuation v are written in red close to the states. For instance, if Player A plays the top row and Player B the left column, the state q_0 is reached. The local interaction \mathcal{F}_{q_0} at state q_0 , valued with the lift μ_v is then depicted in Figure 5. Assume s_A is a Player A positional strategy that is not locally optimal at q_0 : the convex combination of the values in the second column (i.e. the values of the states q_1, q_2) is less than $1/2$, i.e. $p \cdot 1/4 + (1-p) \cdot 3/4 = 1/2 - \varepsilon < 1/2$, where $p \in [0, 1]$ is the probability chosen by $s_A(q_0)$ to play the top row. A Player B strategy s_B whose values at states q_1, q_2 are $(\varepsilon/2)$ -close to $v(q_1), v(q_2)$ and that plays the second row with probability 1 at q_0 , ensures that the value of the game w.r.t. s_A, s_B is at most $p \cdot (1/4 + \varepsilon/2) + (1-p) \cdot (3/4 + \varepsilon/2) \leq 1/2 - \varepsilon/2 < 1/2 = v(q_0)$. Thus, the strategy s_A is not optimal from q_0 .



$$\langle \mathcal{F}_{q_0}, \mu_v \rangle = \begin{bmatrix} 1/2 & 1/4 \\ 1/2 & 3/4 \end{bmatrix}$$

■ **Figure 4** A concurrent game where the values of the states are depicted near them in red. ■ **Figure 5** The local interaction at state q_0 valued by the function v giving the value of states.

However, local optimality is not a sufficient condition for uniform optimality, as it can be seen in Figure 1. A game with a Büchi objective $\text{Büchi}(\{\top\})$ is depicted (Player A wins if the state \top is seen infinitely often). In this game, the valued local interaction at state q_0 is also depicted in Figure 3. A Player A locally optimal strategy plays the top row with probability 1 at state q_0 . If such a strategy is played, Player B can ensure the game never to leave the state q_0 (by playing the left column), thus ensuring value 0 from q_0 (with $v(q_0) = 1$). In fact, in this game, there is no Player A optimal strategy.

Overall, from a state $q \in Q$, the local optimality of a Player A positional strategy s_A ensures that, for every Player B strategy, the convex combination of the values v_H of the ECs H , weighted by the probability to reach them from q , is at least the value of the state q . However, this does not guarantee anything concerning the $\chi_G[s_A]$ -value of the game if it never leaves a specific EC H , it may be 0 whereas $v_H > 0$ (that is what happens in the game of Figure 1). For the strategy s_A to be uniformly optimal, it must ensure that the value under s_A of all states q in the EC H is at least v_H (i.e. for all Player B strategies s_B which ensure staying in H , the value under s_A and s_B is at least v_H). Furthermore, by Theorem 10 applied to H , as soon as at least one state has a value smaller than 1, there is one state with value 0. It follows that, for the strategy s_A to be uniformly optimal, it must be the case that in every EC H such that $v_H > 0$, in the game restricted to H , the value of the game is 1. Note that this exactly corresponds to the condition the authors of [2] have stated in the case of a reachability objective. Uniform optimality can finally be characterized as follows.

► **Lemma 16.** *Assume that W is Borel and prefix-independent, and that in all finite MDPs with objective $Q^\omega \setminus W$, there is a positional optimal strategy. Let $s_A \in \text{PS}_C^A$ be a positional Player A strategy. It is uniformly optimal if and only if:*

- *it is locally optimal;*
- *for all ECs H in the MDP induced by the strategy s_A , if $v_H > 0$, then for all $q \in Q_H$, we have $\chi_{C_H^{s_A}}(q) = 1$ (i.e. in the sub-game formed by the EC H , the value of state q is 1).*

Note that what is proved in the [3] is slightly more general than Lemma 16 as it deals with an arbitrary valuation of the states, not only the one giving the value of the game. This generalization will be used in particular to prove that a positional strategy is ε -optimal from all states in Subsection 5.3.

5 Playing Optimally with Positional Strategies in Büchi Games

In this section, we focus on Büchi objectives. We will distinguish several frameworks. First we consider the case when optimal strategies exist from every state and show that in that case, there is a positional strategy that is uniformly optimal. Furthermore, we show that this always occurs as soon as all local interactions at states not in the target are *reach-maximizable*,

the condition ensuring the existence of optimal strategies in reachability games [2]. Finally, we study the game forms necessary and sufficient to ensure the existence of positional almost-optimal strategies (i.e. ε -optimal strategies, for all $\varepsilon > 0$).

5.1 Playing optimally when optimal strategies exist from every state

First we observe that the value of a Büchi game is closely related to the value of a well-chosen reachability game. We let $\mathcal{G} = \langle \mathcal{C}, \text{Büchi}(T) \rangle$ be a concurrent Büchi game. We modify \mathcal{G} by replacing every state q in the target T , whose value in the Büchi game is $u := \chi_{\mathcal{G}}(q)$ by a state that has probability u to go to \top (the new target in the reachability game) and probability $1 - u$ to go to a sink non-target state \perp . In that case, the value of the original Büchi game is the same as the value of the obtained reachability game, which we denote $\mathcal{G}_{\text{reach}} := \langle \mathcal{C}_{\text{reach}}, \text{Reach}(\{\top\}) \rangle$. It is straightforward to show that the values of both games are the same from all states, and that optimal strategies in the Büchi game will be also optimal in the reachability game. Vice-versa, optimal strategies in the reachability game can be lifted to the Büchi game by augmenting it with a locally optimal strategy at target states.

► **Proposition 17.** *For all Büchi games in which an optimal strategy exists from every state, there exists a Player A positional strategy that is uniformly optimal.*

5.2 Game forms ensuring the existence of optimal strategies

Proposition 17 assumes the existence of optimal strategies from every state. However, there exist Büchi games with no optimal strategies, and even for which almost-optimal strategies require infinite memory. An example of such a game is depicted in Figure 1 (some details will be given in Subsection 5.3). This justifies the interest of having Büchi games in which optimal strategies exist from every state “by design”.

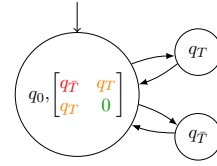
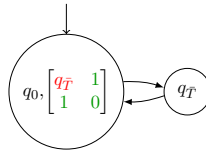
In [2], the authors have proven a necessary and sufficient condition on game forms to ensure the existence of optimal strategies in all finite reachability games using these game forms as local interactions. This condition, called *reach-maximizable* game forms (RM for short) is not detailed here but is available in the [3]. This formalizes as follows:

► **Theorem 18** (Lem 33 and Thm 36 in [2]). *In all reachability games $\mathcal{G} = \langle \mathcal{C}, \text{Reach}(T) \rangle$ where all interactions at states in $Q \setminus T$ are RM, there exist positional uniformly optimal strategies (for Player A). Furthermore, if a game form \mathcal{F} is not RM, one can build a reachability game where \mathcal{F} is the only non-trivial interaction, in which there is no optimal strategy for Player A.*

In the previous subsection, we have described how to translate a Büchi game $\mathcal{G} = \langle \mathcal{C}, \text{Büchi}(T) \rangle$ into a reachability game $\mathcal{G}_{\text{reach}} := \langle \mathcal{C}_{\text{reach}}, \text{Reach}(\{\top\}) \rangle$ while keeping the same value and the same local interactions in states outside the target T ; furthermore, the local interactions in all states in T in $\mathcal{G}_{\text{reach}}$ become trivial (i.e. the outcome is independent of the actions of the players), which are RM. Hence, if all local interactions at states in $Q \setminus T$ in the original game \mathcal{G} are RM, then all game forms in $\mathcal{G}_{\text{reach}}$ are RM, thus there is a uniformly optimal positional strategy for Player A in that game by Theorem 18. Such a strategy can then be translated back into the Büchi game to get a positional Player A uniformly optimal strategy. We obtain the following result.

► **Theorem 19.** *In all Büchi games $\mathcal{G} = \langle \mathcal{C}, \text{Büchi}(T) \rangle$ where all interactions at states in $Q \setminus T$ are RM, there exist positional uniformly optimal strategies (for Player A). Furthermore, if a game form \mathcal{F} is not RM, one can build a Büchi game where \mathcal{F} is the only non-trivial interaction, in which there is no optimal strategy for Player A.*

$$\mathcal{F}_{q_0} = \begin{bmatrix} x & y \\ y & z \end{bmatrix}$$



■ **Figure 6** The game form used as local interaction at q_0 . ■ **Figure 7** The Büchi game \mathcal{G}_1 . ■ **Figure 8** The Büchi game \mathcal{G}_2 .

5.3 GFs ensuring the existence of positional almost-optimal strategies

While positional strategies are sufficient to play optimally in Büchi games where optimal strategies do actually exist, the case is really bad for those Büchi games where optimal strategies do not exist. Indeed, it can be the case that infinite memory is necessary to play almost-optimally, as we illustrate in the next paragraph. Then we characterize game forms that ensure the existence of positional almost-optimal strategies.

An example of a Büchi game where playing almost-optimally requires infinite memory.

Consider the Büchi game $\mathcal{G} = \langle \mathcal{C}, \text{Büchi}(\top) \rangle$ in Figure 1. As argued earlier (in Section 4), there are no optimal strategies in that game. First, notice that the value of the game at state q_0 is 1. However, any finite-memory strategy (i.e. a strategy that can be described with a finite automaton) has value 0. Indeed, consider such a Player A strategy s_A . There is some probability $p > 0$ such that if s_A plays an action with a positive probability, this probability is at least p . If the strategy s_A plays, at state q_0 the bottom row with positive probability and if Player B plays the right column with probability 1, then state \perp is reached with probability at least p . If this happens infinitely often, then the state \perp is eventually reached almost-surely: the value of the strategy is then 0. Hence, Player A has to play, from some time on, the top row with probability 1. From that time on, Player B can play the left column with probability 1, leading to avoid state \top almost-surely. Hence the value of strategy s_A is 0 as well. In fact, all ε -optimal strategies (for every $\varepsilon > 0$) cannot be finite-memory strategies. A Player A strategy with value at least $1 - \varepsilon$ (with $0 < \varepsilon < 1$) will have to play the bottom row with positive probability infinitely often, but that probability will have to decrease arbitrarily close to 0. More details are given in [3].

Game forms ensuring the existence of positional ε -optimal strategies. forms which ensure the existence of positional almost-optimal strategies in Büchi games. The approach is inspired by the one developed in [2] for reachability games.

We start by discussing an example, and then generalize the approach. Let us consider the game form \mathcal{F} depicted in Figure 6, where $\mathcal{O} = \{x, y, z\}$. We embed this game form into two different environments, depicted in Figures 7 and 8. These define two Büchi games using the following interpretation: (a) values 0 and 1 in green represent output values giving the probability to satisfy the Büchi condition when these outputs are selected; (b) other outputs lead to either orange state q_T (a target for the Büchi condition) or red state $q_{\bar{T}}$ (not a target). In particular, the game of Figure 8 is another representation of the game of Figure 1.

Let us compare these two games. First notice that there are no optimal strategies in both cases, as already argued for the game in Figure 8; the arguments are similar for the game of Figure 7. Interestingly, in the game \mathcal{G}_1 in Figure 7, there are positional almost-optimal strategies (it is in fact a reachability game) whereas there are none in the game \mathcal{G}_2 in Figure 8 (as already discussed above); despite the fact that the local interaction at q_0 valued with μ_v , for v the value vector of the game, is that of Figure 3 in both cases.

We analyze the two settings to better understand the differences. A positional ε -optimal strategy s_A at q_0 in both games has to be an ε -optimal GF-strategy $s_A(q_0) \in \mathcal{D}(A)$ in the game in normal form $\langle \mathcal{F}_{q_0}, \mu_v \rangle$ (similarly to how a uniformly optimal positional strategy needs to be locally optimal (Lemma 16)). Consider for instance the Player A positional strategy s_A that plays (at q_0) the top row with probability $1 - \varepsilon$ (which is ε -optimal in $\langle \mathcal{F}_{q_0}, \mu_v \rangle$). This strategy has value $1 - \varepsilon$ in \mathcal{G}_1 , but has value 0 in \mathcal{G}_2 . In both cases, if Player B plays the left column, the target is seen infinitely often almost-surely (since the bottom row is played with positive probability). The difference arises if Player B plays the right column with probability 1: in \mathcal{G}_1 , the target is reached and never left with probability $1 - \varepsilon$; however, in \mathcal{G}_2 , with probability $1 - \varepsilon$, the game visits the target but loops back to q_0 , hence playing for ever this strategy leads with probability 1 to the green value 0. Actually, s_A is ε -optimal if for each column, either there is no green value but at least one orange q_T , or there are green values and their average is at least $1 - \varepsilon$.

This intuitive explanation can be generalized to any game form \mathcal{F} as follows, which we will embed in several environments. To define an environment, we fix (i) a partition $O = O_{Lp} \uplus O_{Ex}$ of the outcomes (Lp stands for loop – i.e. orange and red outcomes – and Ex stands for exit – i.e. the green outcomes), (ii) a partial valuation of the outcomes $\alpha : O_{Ex} \rightarrow [0, 1]$ and a probability $p_T : O_{Lp} \rightarrow [0, 1]$ to visit the target T in the next step and loop back (above, the probability was either 1 (represented by orange state q_T) or 0 (represented by red state $q_{\bar{T}}$)). We then consider the Büchi game $\mathcal{G}_{\mathcal{F}, \alpha, p_T}^{\text{Büchi}}$ which embeds game form \mathcal{F} in the environment given by α and p_T as follows: an outcome $o \in O_{Lp}$ leads to q_T with probability $p_T(o)$ and $q_{\bar{T}}$ otherwise; from q_T or $q_{\bar{T}}$, surely we go back to q_0 ; an outcome $o \in O_{Ex}$ leads to the target T with probability $\alpha(o)$ and outside T otherwise (in both cases, it stays there forever); this is formally defined in [3]. We want to specify that there are positional ε -optimal strategies in the game $\mathcal{G}_{\mathcal{F}, \alpha, p_T}^{\text{Büchi}}$ if and only if there are ε -optimal GF-strategies in the game form \mathcal{F} ensuring the properties described above. However, to express what is an ε -optimal strategy, we need to know the value of the game $\mathcal{G}_{\mathcal{F}, \alpha, p_T}^{\text{Büchi}}$ at state q_0 : to do so, we use [11], in which the value of concurrent games with parity objectives (generalizations of Büchi and co-Büchi objectives) is computed using μ -calculus. In our case, this can be expressed with nested fixed point operations, as described in the Appendix of [3]. We denote this value $u_{\mathcal{F}, \alpha, p_T}^{\text{Büchi}}$ or simply u . We can now define almost-Büchi maximizable (aBM for short) game form.

► **Definition 20** (Almost-Büchi maximizable game forms). *Consider a game form \mathcal{F} , a partition of the outcomes $O = O_{Lp} \uplus O_{Ex}$, a partial valuation of the outcomes $\alpha : O_{Ex} \rightarrow [0, 1]$ and a probability $p_T : O_{Lp} \rightarrow [0, 1]$ to visit the target T . The game form \mathcal{F} is almost-Büchi maximizable (aBM for short) w.r.t. α and p_T if for all $0 < \varepsilon < u$, there exists a GF-strategy $\sigma_A \in \mathcal{D}(\text{St}_A)$ such that, for all $b \in \text{St}_B$, letting $A_b := \{a \in \text{Supp}(\sigma_A) \mid \varrho(a, b) \in O_{Ex}\}$, either:*

- $A_b = \emptyset$, and there exists $a \in \text{Supp}(\sigma_A)$ such that $p_T(\varrho(a, b)) > 0$ (i.e. if all outcomes loop back to q_0 , there is a positive probability to visit T : left column in the game \mathcal{G}_2);
- $A_b \neq \emptyset$, and $\sum_{a \in A_b} \sigma_A(a) \cdot \alpha(\varrho(a, b)) \geq (u - \varepsilon) \cdot \sigma_A(A_b)$ (i.e. for the action b , the value of σ_A restricted to the outcomes in O_{Ex} is at least $u - \varepsilon$: right column in the game \mathcal{G}_1).

The game form \mathcal{F} is almost-Büchi maximizable (aBM for short) if for all partitions $O = O_{Lp} \uplus O_{Ex}$, for all partial valuations $\alpha : O_{Ex} \rightarrow [0, 1]$ and probability function $p_T : O_{Lp} \rightarrow [0, 1]$, it is aBM w.r.t. α and p_T .

This definition relates to the existence of positional almost-optimal strategies in $\mathcal{G}_{\mathcal{F}, \alpha, p_T}^{\text{Büchi}}$:

► **Lemma 21.** *The game form \mathcal{F} is aBM w.r.t. α and p_T if and only if there are positional almost-optimal strategies from state q_0 in the game $\mathcal{G}_{\mathcal{F}, \alpha, p_T}^{\text{Büchi}}$.*

33:12 Playing (Almost-)Optimally in Concurrent Büchi and Co-Büchi Games

More interestingly, if a concurrent Büchi game has all its game forms aBM (possibly except at target states), then there exist positional almost-optimal strategies.

► **Theorem 22.** *Consider a Büchi game $\mathcal{G} = \langle \mathcal{C}, \text{Büchi}(T) \rangle$ and assume that all local interactions at states in $Q \setminus T$ are aBM. Then, for every $\varepsilon > 0$, there is a positional strategy that is ε -optimal from every state $q \in Q$.*

Proof sketch. Let $\varepsilon > 0$. We build a positional Player A strategy s_A and then apply (a slight generalization of) Lemma 16 to show that it is ε -optimal. Let $v := \chi_{\mathcal{G}}$ be the value vector of the game and $u \in v[Q] \setminus \{0\}$ be some positive value of the game. Consider the set $Q_u := v^{-1}[\{u\}] \subseteq Q$ of states whose values w.r.t. v is u . We define the strategy s_A on each Q_u for $u \in v[Q]$ and then glue the portions of s_A together. This is possible because the Player A strategy we build enforces end-components in which the value given by v of all states is the same (similarly to Proposition 15 for locally optimal strategies).

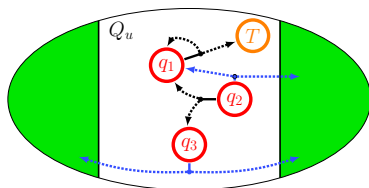
In Figure 9, the set Q_u corresponds to the white area. Target states (T) are in orange, while non-target states are in red (q_1, q_2, q_3 in the figure). From every state of Q_u , there may be several split arrows, which correspond to choices by Player B (actions $b \in \text{St}_B$); black split edges stay within Q_u while blue edges partly lead outside Q_u ; once a split edge is chosen by Player B, Player A may ensure any leaving edge with some positive probability.

In a state $q \in Q_u \cap T$, the strategy s_A only needs to be locally optimal, i.e. such that $\text{val}_{\langle \mathcal{F}_q, \mu_v \rangle}(s_A(q)) = v(q)$. Then, the states in $Q_u \setminus T$ will be considered one by one. Let $q \in Q_u \setminus T$, we consider the Büchi game $\mathcal{G}_{\mathcal{F}_q, \alpha, p_T}^{\text{Büchi}}$ built from the aBM game form \mathcal{F}_q and the immediate environment of q as follows: the partial valuation α of the outcomes (i.e. the Nature states) is defined on those with a positive probability to reach $Q \setminus Q_u$ (i.e. the green area – green outcomes in Figures 7, 8), and p_T maps a Nature state d staying in Q_u to the probability $\text{dist}(d)[Q_u \cap T]$ to reach the target T (i.e. the probability to reach the orange states in Figure 9 – they correspond to the orange and red outcomes in Figures 7, 8).

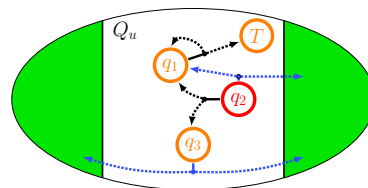
In Figure 9, states in red are non-winning yet (non-target in the first stage) and therefore if Player B can choose a black split-edge leading only to red states, then the value of game $\mathcal{G}_{\mathcal{F}_q, \alpha, p_T}^{\text{Büchi}}$ is 0 (this is the case of q_2). On the other hand, if all split-edges are either blue or black with at least one orange end, then the value of game $\mathcal{G}_{\mathcal{F}_q, \alpha, p_T}^{\text{Büchi}}$ is positive (this is the case of states q_1 and q_3). Then, we realize that there must be at least one state $q \in Q_u \setminus T$ such that $u_{\mathcal{F}_q, \alpha, p_T}^{\text{Büchi}} \geq u$ (this is a key argument, and it is due to the definition of $u_{\mathcal{F}_q, \alpha, p_T}^{\text{Büchi}}$, which relates to how the value in Büchi games is computed via fixed-point operations). In Figure 9, there are actually two such states, q_1 and q_3 : the value at q_1 is 1 while the value at q_3 is an average of the values at the two (green) ends of the (blue) split-edge. We can then use the facts that \mathcal{F}_{q_1} and \mathcal{F}_{q_3} are aBM and apply the definition with a well-chosen $0 < \varepsilon_u \leq \varepsilon$ to obtain the GF-strategies played by the strategy s_A at states q_1 and q_3 .

We then iterate the process by going to the second stage by considering that the previously dealt states (q_1 and q_3 in our example) are now orange, as in Figure 10: they are now considered as targets. The property that $u_{\mathcal{F}_q, \alpha, p_T}^{\text{Büchi}} \geq u$ (with a new p_T taking into account the larger set of targets) then propagates throughout the game to all states in the white area. The strategy s_A is now fully defined on Q_u , and we can check that, under any Player B strategy, if the game eventually stays within an EC within Q_u , it will reach the target T infinitely often almost-surely. Indeed, from each state, there is either a positive probability to leave the EC (i.e. see a green outcome) or a positive probability to get closer to the target (i.e. see an orange outcome). ◀

With Lemma 21 and Theorem 22, we obtain a corollary analogous to Theorem 19 for aBM game forms and the existence of almost-optimal strategies.



■ **Figure 9** First stage of the construction of strategy s_A within Q_u .



■ **Figure 10** Second stage of the construction of strategy s_A within Q_u .

► **Corollary 23.** *In all Büchi games $\mathcal{G} = \langle C, \text{Büchi}(T) \rangle$ where all interactions at states in $Q \setminus T$ are aBM, there exist positional uniformly almost-optimal strategies (for Player A). Furthermore, if a game form \mathcal{F} is not aBM, one can build a Büchi game where \mathcal{F} is the only non-trivial interaction, in which there is no positional almost-optimal strategy for Player A.*

Note that game forms that are not aBM may behave well in some environments, even though they do not behave well in some other environments. Hence, it might be the case that in a specific Büchi game with local interactions that are not aBM, there are some positional almost-optimal strategies. This observation stands for all the type of game forms we have characterized: RM, aBM and also coBM in the next section.

Finally, note that it is decidable whether a game form is aBM.

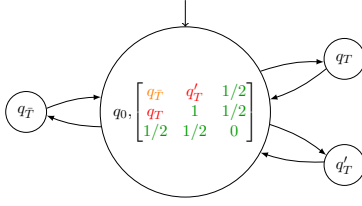
► **Proposition 24.** *It is decidable if a game form is aBM. Moreover, all RM game forms are aBM game forms and there is a game form that is aBM but not RM.*

6 Playing Optimally in co-Büchi Games

Although they may seem quite close, Büchi and co-Büchi objectives do not enjoy the same properties in the setting of concurrent games. For instance, we have seen that there are Büchi games in which a state has value 1 but any finite-memory strategy has value 0. This cannot happen in co-Büchi games, since strategies with values arbitrarily close to 1 can be found among positional strategies [5]. We have also seen in Section 5 that in all concurrent Büchi games, if there is an optimal strategy from all states, then there is a uniformly optimal positional strategy. As we will see in the next subsection, this does not hold for co-Büchi games. This shows that concurrency in games complicates a lot the model: the results of this paper has to be put in regards with the model of turn-based games where pure positional strategies are sufficient to play optimally, for parity objectives [15].

6.1 Optimal strategies may require infinite memory in co-Büchi games

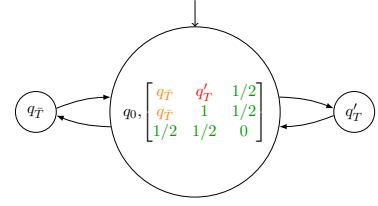
Consider the game depicted in Figure 11, which uses the same convention as in the previous section. The objective is $W = \text{coBüchi}(\{q_T, q'_T, \top\})$ where the state \top is not represented, but implicitly present via the green values (for instance, green value $1/2$ leads to \top with probability $1/2$ and to \perp with probability $1/2$). Let $A := \{a_1, a_2, a_3\}$ with a_1 the top row and a_3 the bottom row and $B := \{b_1, b_2, b_3\}$ with b_1 the leftmost column and b_3 the rightmost column. If a green value is not reached, Player A wins if and only if eventually, the red state is not seen anymore. The values of the states $q_0, q_T, q_{T'}$ and $q_{\bar{T}}$ are the same and are at most $1/2$. Indeed, if Player B almost-surely plays b_3 at q_0 , she ensures that the value of the game from q_0 is at most $1/2$. Let us argue that any Player A positional strategy has value less than $1/2$ and exhibit an infinite-memory Player A strategy whose value is $1/2$.



■ **Figure 11** The co-Büchi game \mathcal{G}_1 .

$$\mathcal{F} = \begin{bmatrix} x & z & t \\ y & r & t \\ t & t & s \end{bmatrix}$$

■ **Figure 12** The local interaction at state q_0 .



■ **Figure 13** Another co-Büchi game (\mathcal{G}_2) built on \mathcal{F} .

Consider a Player A positional strategy s_A . We define a Player B strategy s_B as follows: if $s_A(q_0)(a_3) = \varepsilon > 0$, then we set $s_B(q_0)(b_3) := 1$ and the value of the game w.r.t. s_A, s_B is at most $1/2 - \varepsilon < 1/2$. If $s_A(q_0)(a_1) = 1$, we set $s_B(q_0)(b_2) := 1$ and the state $q'_T \in T$ is visited infinitely often almost-surely. Otherwise, $s_A(q_0)(a_2) > 0$ and $s_A(q_0)(a_3) = 0$, hence choosing $s_B(q_0)(b_1) := 1$ ensures that the state $q_T \in T$ is visited infinitely often almost-surely. In the last two cases, s_A has value 0. Overall, any positional strategy s_A has value less than $1/2$.

We briefly describe a Player A optimal strategy s_A (whose value is $1/2$). The idea is the following: along histories that have not visited q'_T yet (this happens when Player B has not played b_2), s_A plays a_1 with very high probability $1 - \varepsilon_k < 1$ and a_2 with probability $\varepsilon_k > 0$, where k denotes the number of steps. The values $(\varepsilon_k)_{k \in \mathbb{N}}$ are chosen so that, if Player B only plays b_1 with probability 1, then the state $q_T \in T$ is seen finitely often almost-surely. Some details are given in Appendix B. After the first visit to q'_T , Player s_A switches to a positional strategy of value $\frac{1}{2} - \varepsilon'_k$, for $\varepsilon'_k > 0$ and k is the number of steps after the first visit to q'_T . A first visit to q'_T occurs when b_2 is played by Player B. The value of s_A after that point is then $(1 - \varepsilon_k) \cdot (\frac{1}{2} - \varepsilon'_k) + \varepsilon_k \cdot 1$. It suffices to choose ε'_k small enough so that the above value is at least $1/2$. Such a Player A strategy is optimal from q_0 . It follows that, contrary to the Büchi case, requiring that positional optimal strategies exists from all states in a co-Büchi game is stronger than requiring that optimal strategies exists from all states.

6.2 GFs in co-Büchi games which ensure positional optimal strategies

Contrary to the Büchi objectives, RM game forms do not suffice to ensure the existence of positional uniformly optimal strategies in co-Büchi games, see Proposition 26. In this subsection, we characterize the game forms ensuring this property.

We proceed as in Subsection 5.3, and we explain the approach on an example. Consider the game form \mathcal{F} depicted in Figure 12, that is the local interaction at state q_0 of the game in Figure 11. Let $O = O_{E_x} \uplus O_{L_p}$ for $O_{E_x} := \{t, r, s\}$, $O_{L_p} := \{x, y, z\}$ and consider the partial valuation $\alpha : O_{E_x} \rightarrow [0, 1]$ such that $\alpha(t) := 1/2$, $\alpha(r) := 1$ and $\alpha(s) := 0$. We then consider two probability functions $p_T^1, p_T^2 : O_{L_p} \rightarrow [0, 1]$ such that $p_T^1(x), p_T^2(x) := 0$, $p_T^1(z), p_T^2(z) := 1$, $p_T^1(y) := 1$ whereas $p_T^2(y) := 0$. The games $\mathcal{G}_1 := \mathcal{G}_{\mathcal{F}, \alpha, p_T^1}^{\text{coBüchi}}$ (Figure 11) and $\mathcal{G}_2 := \mathcal{G}_{\mathcal{F}, \alpha, p_T^2}^{\text{coBüchi}}$ (Figure 13) are defined just like their Büchi counterparts, except for the objective which is $W = \text{coBüchi}(T)$ with $q_T, q'_T \in T$ and $q''_T \notin T$. We have already argued that there is no positional optimal strategy in the game \mathcal{G}_1 (Figure 11). The issue is the following: when considering a locally optimal strategy (recall Lemma 16), there is a column where, in the support of the strategy, there is a red outcome (i.e. positive probability to reach T) and no green outcome. Then, if Player B plays, with probability 1, the corresponding action, she ensures that the set T is visited infinitely often almost-surely. Now, in the game \mathcal{G}_2 of Figure 12, any Player A positional strategy playing a_3 with probability 0 and a_2 with positive probability is uniformly optimal. Indeed, in that case, (i) if b_1 is played, then the set T is not seen; (ii) if b_2 or b_3 are played, then the game will end in a green outcome almost-surely.

■ **Table 1** Game forms necessary and sufficient for the existence of positional strategies for (almost-)optimality.

	Positional Optimal Strategy		Positional ε -Optimal Strategy	
	T	$Q \setminus T$	T	$Q \setminus T$
Safe(T)	All	All	All	All
Reach(T)	All	RM	All	All
Büchi(T)	All	RM	All	aBM
coBüchi(T)	RM	coBM	All	All

In the general case, as for Büchi games, the value of co-Büchi games can be computed with nested fixed points (see [3]). Using this value, we can define the notion of co-Büchi maximizable game forms (coBM for short), which, while requiring a slightly more complex setting, formalizes the intuition given in the previous example, see [3]. It ensures a lemma analogous to Lemma 21 in the context of co-Büchi games.

Furthermore, coBM game forms also ensure that, when they are used in a co-Büchi game, there always exists a positional uniformly optimal strategy.

► **Theorem 25.** *In all co-Büchi games $\mathcal{G} = \langle \mathcal{C}, \text{coBüchi}(T) \rangle$ where all local interactions at states in T are RM and all local interactions at states in $Q \setminus T$ are coBM, there exist positional uniformly optimal strategies (for Player A). Furthermore, if a game form \mathcal{F} is not coBM, one can build a co-Büchi game where \mathcal{F} is the only non-trivial interaction where there is no positional optimal strategy for Player A.*

► **Proposition 26.** *It is decidable if a game form is coBM. All coBM game forms are RM game forms. There exists a game form that is RM but not coBM.*

7 Conclusion

We have studied game forms and defined various conditions such that these game forms in isolation behave properly w.r.t. some fixed property (like existence of optimal strategies for Büchi objectives), and we have proven that they can be used collectively in graph games while preserving this property. These conditions, summarized in Table 1, give the unique way to construct games which will satisfy good memory properties by construction for playing (almost-)optimally.

Let us explicit how to read a specific row of this table, say the third one for the Büchi objective. A Büchi objective is defined along with a target $T \subseteq Q$. The game forms necessary and sufficient to ensure the existence of positional optimal strategies are given in the leftmost part of the table, and to ensure the existence of positional almost-optimal strategies, in the rightmost part of the table. For instance, for the leftmost part, if a game form is not RM, there is a Büchi game built from it – where it is the only non-trivial local interaction – in which there is no positional optimal strategy. Conversely, if in a Büchi game, all local interactions at states outside the target T are RM game forms (no further assumption is made on the game forms appearing at states in T), then there is a positional optimal strategy. The rightmost part of the table can be read similarly.

Finally, we would like to mention that all two-variable game forms $\mathcal{F} = \langle \text{St}_A, \text{St}_B, \text{O}, \varrho \rangle$ (i.e. such that $|\text{O}| \leq 2$) are coBM (this is a direct consequence of the definition), hence RM. From this, we obtain as a corollary of our results that in all finite Büchi and co-Büchi games where all local interactions are two-variable game forms, both players have positional uniformly optimal strategies.

► **Corollary 27.** *In a Büchi or co-Büchi game \mathcal{G} such that for all $q \in Q$, $|\delta(q, A, B)| \leq 2$, both players have a positional uniformly optimal strategy.*

References

- 1 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Handbook of Model Checking*, chapter Graph games and reactive synthesis, pages 921–962. Springer, 2018.
- 2 Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Optimal strategies in concurrent reachability games. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 7:1–7:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CSL.2022.7.
- 3 Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Playing (almost-)optimally in concurrent büchi and co-büchi games. *CoRR*, abs/2203.06966, 2022. doi:10.48550/arXiv.2203.06966.
- 4 Krishnendu Chatterjee. Concurrent games with tail objectives. *Theor. Comput. Sci.*, 388(1-3):181–198, 2007. doi:10.1016/j.tcs.2007.07.047.
- 5 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 Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 678–687. ACM Press, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109631>.
- 6 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Qualitative concurrent parity games. *ACM Trans. Comput. Log.*, 12(4):28:1–28:51, 2011. doi:10.1145/1970398.1970404.
- 7 Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Quantitative stochastic parity games. In *Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 121–130. SIAM, 2004.
- 8 Luca de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
- 9 Luca de Alfaro, Thomas Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.
- 10 Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 141–154. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855763.
- 11 Luca de Alfaro and Rupak Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.
- 12 Marta Kwiatkowska, Gethin Norman, Dave Parker, and Gabriel Santos. Automatic verification of concurrent stochastic systems. *Formal Methods in System Design*, 58:188–250, 2021. doi:10.1007/s10703-020-00356-y.
- 13 Marta Kwiatkowska, Gethin Norman, David Parker, Gabriel Santos, and Rui Yan. Probabilistic model checking for strategic equilibria-based decision making. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.MFCS.2022.4.
- 14 Donald A. Martin. The determinacy of blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.
- 15 Annabelle McIver and Carroll Morgan. Games, probability and the quantitative μ -calculus $qm\mu$. In *Proc. 9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, volume 2514 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2002.
- 16 Wolfgang Thomas. Infinite games and verification. In *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–64. Springer, 2002. Invited Tutorial.
- 17 John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton Univ. Press, Princeton, 1944.

- 18 Wiesław Zielonka. Perfect-information stochastic parity games. In *Proc. 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'04)*, volume 2987 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2004.

A Infinite memory to play ε -optimal strategies in Büchi games

Consider the game of Figure 1. This game is explained in [10]. Assume that the sets of actions are $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$ with a_1 corresponding to the top row and b_1 to the left column.

Consider some positive probability $p > 0$ and consider a Player A strategy s_A such that the probability to play a_2 is either 0 or at least p . Let us show that the value of such a strategy is 0. Specifically, consider a Player B strategy s_B such that, for all $\rho \cdot q_0 \in Q^+$, we have:

$$s_B(\rho \cdot q_0) := \begin{cases} b_1 & \text{if } s_A(\rho \cdot q_0)(a_2) = 0 \\ b_2 & \text{if otherwise} \end{cases}$$

Each time $s_A(\rho \cdot q_0)(a_2) \geq p$, then the probability to reach the state \perp is reached with probability at least p . Hence, if this happens infinitely often, then the state \perp is seen with probability 1. Otherwise, the state q_0 is never left after some moment on. In both cases, the set T is seen only finitely often. It follows that the value of the game with strategies s_A, s_B is 0.

Now, consider some $\varepsilon > 0$ let us exhibit a Player A strategy of value at least $1 - \varepsilon$. The idea is the following. Consider a sequence $(\varepsilon_k)_{k \in \mathbb{N}}$ of positive values such that $\lim_{n \rightarrow \infty} \prod_{i=0}^n (1 - \varepsilon_i) \geq 1 - \varepsilon$. Then, consider a Player A strategy such that $s_A(\rho)(a_1) := 1 - \varepsilon_k$ where $k \in \mathbb{N}$ denotes the number of times the state \top is visited in ρ . Then, the state q_0 is seen indefinitely with probability 0. If the state \top has been seen already k times, then the probability to stay at state q_0 for n steps is at most $(1 - \varepsilon_k)^n \rightarrow_{n \rightarrow \infty} 0$. Furthermore, the probability to ever reach the state \perp is at most $\lim_{n \rightarrow \infty} \prod_{i=0}^n (1 - \varepsilon_i) \geq 1 - \varepsilon$. Overall, regardless of Player B's strategy, the probability to visit the set T infinitely often is at least $1 - \varepsilon$. Note that such a sequence $(\varepsilon_k)_{k \in \mathbb{N}}$ could be equal, for instance, to $\varepsilon_k := 1 - (1 - \varepsilon)^{\frac{1}{2^{k+1}}}$. Then, for $n \in \mathbb{N}$:

$$\prod_{i=0}^n (1 - \varepsilon_i) = (1 - \varepsilon) \sum_{i=0}^n \frac{1}{2^{i+1}} = (1 - \varepsilon)^{(1 - \frac{1}{2^{n+1}})} \rightarrow_{n \rightarrow \infty} 1 - \varepsilon$$

B Playing optimally in the game of Figure 11

We are given a probabilistic process such that, at each step either event T or $\neg T$ occur. Furthermore, at step $n \in \mathbb{N}$, the probability that the event T occurs is equal to ε_n . We will denote by \mathbb{P} the probability measure of the corresponding measurable sets. In the following, we will use the following notations:

- for all $n \in \mathbb{N}$, $X_n T$ refers to the event: the event T occurs at step n .
- for all $n \in \mathbb{N}$: $\diamond_n T := \cup_{k \geq n} (X_k T)$ refers to the event: the event T occurs at some point after step n .
- $\square \diamond T := \cap_{n \in \mathbb{N}} (\diamond_n T)$ refers to the event: the event T occurs infinitely often.

An infinite path (which can be seen as an infinite sequence of elementary events) is winning for the Büchi objective if it corresponds to the event $\square \diamond T$. We want to define a sequence $(\varepsilon_k)_{k \in \mathbb{N}}$ such that, for all $k \in \mathbb{N}$ we have $\varepsilon_k > 0$ and $\mathbb{P}(\square \diamond T) = 0$. In fact, it suffices to consider a sequence $(\varepsilon_k)_{k \in \mathbb{N}}$ whose sum converges (i.e. such that $\sum_{n=0}^{\infty} \varepsilon_n < \infty$), for instance $\varepsilon_n := \frac{1}{2^{n+1}}$ for all $n \in \mathbb{N}$. Indeed, for all $n \in \mathbb{N}$, we have:

33:18 Playing (Almost-)Optimally in Concurrent Büchi and Co-Büchi Games

$$\mathbb{P}(X_n T) = \varepsilon_n$$

Furthermore:

$$\mathbb{P}(\diamond_n T) = \mathbb{P}(\cup_{k \geq n} X_k T) \leq \sum_{k=n}^{\infty} \mathbb{P}(X_k T) = \sum_{k=n}^{\infty} \varepsilon_k$$

It follows that:

$$\mathbb{P}(\square \diamond T) = \mathbb{P}(\cap_{n \in \mathbb{N}} \diamond_n T) = \lim_{n \rightarrow \infty} \mathbb{P}(\diamond_n T) \leq \lim_{n \rightarrow \infty} \sum_{k=n}^{\infty} \varepsilon_k = 0$$

Ambiguity Through the Lens of Measure Theory

Olivier Carton 

IRIF, Université Paris Cité & CNRS, France

Abstract

In this paper, we establish a strong link between the ambiguity for finite words of a Büchi automaton and the ambiguity for infinite words of the same automaton. This link is based on measure theory. More precisely, we show that such an automaton is unambiguous, in the sense that no finite word labels two runs with the same starting state and the same ending state if and only if for each state, the set of infinite sequences labelling two runs starting from that state has measure zero. The measure used to define these negligible sets, that is sets of measure zero, can be any measure computed by a weighted automaton which is compatible with the Büchi automaton. This latter condition is very natural: the measure must only put weight on sets $wA^{\mathbb{N}}$ where w is the label of some run in the Büchi automaton.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases ambiguity, Büchi automata, measure theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.34

Related Version *Full Version:* <https://arxiv.org/abs/2011.10534>

Funding Work supported by ANR CODYS (ANR-18-CE40-0007).

1 Introduction

The relationship between deterministic and non-deterministic machines has been extensively studied since the very beginning of computer science. Despite these efforts, many questions remain wide open. This is of course true in complexity theory for questions like P versus NP but also in automata theory [15, 12]. It is for instance not known whether the simulation of non-deterministic either one-way or two-way automata by deterministic two-way automata requires an exponential blow-up of the number of states [20].

Unambiguous machines are usually defined as non-deterministic machines in which each input has at most one accepting run. They are intermediate machines in between the two extreme cases of deterministic and non-deterministic machines. The notion of ambiguity considered in the paper is slightly stronger and more structural as it does not depend on initial and final states. In the case of automata accepting finite words, non-deterministic automata can be exponentially more succinct than unambiguous automata which can be, in turn, exponentially more succinct than deterministic automata [22]. However, the inclusion problem for unambiguous automata is tractable in polynomial time [23] like for deterministic automata while the same problem for non-deterministic automata is PSPACE-complete [1, Section 10.6].

The polynomial time algorithm for the inclusion of unambiguous automata accepting finite words in [23] is based on a clever counting argument which cannot easily be adapted to infinite words. It is still unknown whether the inclusion problem for unambiguous Büchi automata can be solved in polynomial time. The problem was solved in [17] for sub-classes of Büchi automata with weak acceptance conditions and in [6] for prophetic Büchi automata introduced in [10] (see also [19, Sec. II.10]) which are strongly unambiguous. These latter results are obtained through reductions of the problem for infinite words to the problem for finite words. The main result of this paper can be seen as a first step towards a solution for all Büchi automata as it connects ambiguity for infinite words to ambiguity for finite words and thus provides a better understanding of ambiguity for infinite words.



© Olivier Carton;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 34; pp. 34:1–34:14



Leibniz International Proceedings in Informatics

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

The aim of this paper is to exhibit a strong link between the ambiguity of some automaton for finite words and the ambiguity of the same automaton for infinite words. The paper is focused on strongly connected Büchi automata. Two examples given in the conclusion show that the problem is more involved for non strongly connected automata. It turns out that unambiguity for infinite words implies the unambiguity for finite words but the converse does not hold in general. This converse can however be recovered if unambiguity for infinite words is considered up to a negligible set of inputs. *Negligible* should here be understood as a set of zero measure. The measure used to characterize ambiguity must fulfill some compatibility conditions with the automaton. Some examples show these conditions cannot be avoided. Note that measures were already used to characterize maximal variable-length codes [3, Thm. 5.10] which are, in essence, a combinatorial definition of non-ambiguity.

The first step of the proof is to show that the measure of the set of accepted sequences does not increase if all states of the automaton are made final. This result is interesting by itself but it also reduces the proof of our result to automata with all states final. Since initial states are also not relevant, the problem is again reduced to automata with all states initial and final, which accept the so called shift spaces from symbolic dynamics [18]. This special case is handled using techniques from this domain like synchronizing words and Fisher covers.

This work was motivated by questions about automata with outputs also known as transducers. These transducers realize functions mapping infinite sequences to infinite sequences and the questions are focused on the long term behaviour. A natural question is the preservation of normality where normality is the property that all blocks of the same length occur with the same limiting frequency [9]. Normality was introduced by Borel to formalize the most basic form of randomness for real numbers [5]. It turns out that normality can be characterized by non-compressibility by transducers realizing one-to-one functions [2]. Since each infinite run ends in a strongly connected component, it is sufficient to study ambiguity of strongly connected automata. It is a classical result that each function realized by a transducer can be realized by a transducer whose input automaton is unambiguous [11]. The result proved in this paper shows that if all states of a strongly connected unambiguous transducer are made final the transducer remains unambiguous up to a set of measure zero. It allows us to use, for instance, the ergodic theorem for Markov chains where the function must be defined up to a set of measure zero.

The paper is organized as follows. Section 2 is devoted to basic definitions needed for the main result which is stated in Section 3. The first step of the proof is to reduce the problem to the special case of Büchi automata with all states final. This is done in Section 4. The proof of this special case is carried out in Section 5.

2 Definitions

2.1 Words, sequences and measures

Let A be a finite set of *symbols* that we refer to as the *alphabet*. We write $A^{\mathbb{N}}$ for the set of all sequences on the alphabet A and A^* for the set of all (finite) words. The length of a finite word w is denoted by $|w|$. The positions of sequences and words are numbered starting from 1. The empty word is denoted by ε . The cardinality of a finite set E is denoted by $\#E$. A *factor* of a sequence $a_1a_2a_3\cdots$ is a finite word of the form $a_k a_{k+1} \cdots a_{\ell-1}$ for integers $1 \leq k \leq \ell$ where $k = \ell$ yields the empty word ε . We let $\text{fact}(X)$ denote the set of factors of a set X of sequences.

We recall here a few notions of topology. The set $A^{\mathbb{N}}$ of sequences can be endowed with a topology by the distance d which is defined as follows. The distance $d(x, y)$ of two sequences $x = a_1a_2a_3\cdots$ and $y = b_1b_2b_3\cdots$ is zero if $x = y$ and is $2^{-\min\{i:a_i \neq b_i\}}$ otherwise. The set X

is *open* if it is equal to a possibly infinite union of *cylinders*, that is, sets of the form $wA^{\mathbb{N}}$ for $w \in A^*$. It is *closed* if its complement in $A^{\mathbb{N}}$ is open. Each set X is contained in a smallest closed set \bar{X} called its *closure*. The complement of \bar{X} is the union of all sets $wA^{\mathbb{N}}$ which are disjoint from X .

By measure, we mean, in this paper, a *probability measure on $A^{\mathbb{N}}$* , that is, a σ -additive set function μ which assigns to each Borelian set $X \subseteq A^{\mathbb{N}}$ a real number $\mu(X)$ (called its measure) in the interval $[0, 1]$ and which satisfies $\mu(A^{\mathbb{N}}) = 1$. The σ -additivity property means that if X_0, X_1, X_2, \dots is a collection of pairwise disjoint sets, then

$$\mu\left(\bigcup_{n \geq 0} X_n\right) = \sum_{n \geq 0} \mu(X_n).$$

Most sets considered in this paper are rational, and therefore, they are Borelian, (in the level Δ_3^0 of the Borel hierarchy) and their measure does always exist. By the Carathéodory Extension Theorem, each measure μ is fully determined by the measures of the cylinders sets, that is, sets of the form $wA^{\mathbb{N}}$ for some finite word w . In the rest of the paper, we identify a measure μ on $A^{\mathbb{N}}$ and the function which maps each finite word w to the measure $\mu(wA^{\mathbb{N}})$ of the cylinder set $wA^{\mathbb{N}}$.

A *probability measure on A^** is a function $\mu : A^* \rightarrow [0, 1]$ such that $\mu(\varepsilon) = 1$ and that the equality

$$\sum_{a \in A} \mu(wa) = \mu(w)$$

holds for each word $w \in A^*$. The simplest example of a probability measure is a *Bernoulli measure*. It is a monoid morphism from A^* to $[0, 1]$ (endowed with multiplication) such that $\sum_{a \in A} \mu(a) = 1$. Among the Bernoulli measures is the *uniform measure* which maps each word $w \in A^*$ to $(\#A)^{-|w|}$. In particular, each symbol a is mapped to $\mu(a) = 1/\#A$.

By the Carathéodory Extension Theorem, a measure μ on A^* can be uniquely extended to a probability measure $\hat{\mu}$ on $A^{\mathbb{N}}$ such that $\hat{\mu}(wA^{\mathbb{N}}) = \mu(w)$ holds for each word $w \in A^*$. As already said, we use the same symbol for μ and $\hat{\mu}$. A probability measure μ is said to be (*shift*) *invariant* if the equality

$$\sum_{a \in A} \mu(aw) = \mu(w)$$

holds for each word $w \in A^*$. The support $\text{supp}(\mu)$ of a measure μ is the set $\text{supp}(\mu) = \{w \in A^* : \mu(w) > 0\}$ of finite words.

The column vector such that each of its entries is 1 is denoted by $\mathbf{1}$. A P -vector λ is called *stochastic* (respectively, *substochastic*) if its entries are non-negative and sum up to 1 (respectively, to at most 1). that is, $0 \leq \lambda_p \leq 1$ for each $p \in P$ and $\lambda\mathbf{1} = 1$ (respectively, $\lambda\mathbf{1} \leq 1$). A matrix M is called *stochastic* (respectively, *substochastic*) if each of its rows is stochastic (respectively, *substochastic*), that is $M\mathbf{1} = \mathbf{1}$ (respectively, $M\mathbf{1} \leq \mathbf{1}$). It is called *strictly substochastic* if it is substochastic but not stochastic. This means that the entries of at least one of its rows sum up to a value which is strictly smaller than 1.

In the paper, we mainly consider rational measures also known as hidden Markov measures and under other names in the literature [7]. These measures are those for which the values $\mu(w)$ for $w \in A^*$ are given by a weighted automaton [21, Chap .4] or equivalently by a (matrix) representation. A measure μ is *rational* if there is an integer m , a row $(1 \times m)$ -vector π , a morphism ν from A^* into $m \times m$ -matrices over real numbers and a column $(m \times 1)$ -vector ρ such that the following equality holds for each word $a_1 \cdots a_k$ [4].

$$\mu(a_1 \cdots a_k) = \pi \nu(a_1 \cdots a_k) \rho = \pi \nu(a_1) \cdots \nu(a_k) \rho$$

The triple $\langle \pi, \nu, \rho \rangle$ is called a *representation* of the rational measure μ and the integer m the *dimension*. By the main result in [14], it can always be assumed that both the vector π and the matrix $\sum_{a \in A} \nu(a)$ are stochastic and that the vector ρ is the vector $\mathbf{1}$. The triple $\langle \pi, \nu, \mathbf{1} \rangle$ is then called a *stochastic representation* of μ . Note that if the representation $\langle \pi, \nu, \rho \rangle$ is stochastic, the function μ defined by $\mu(w) = \pi \nu(w) \rho$ is a measure because it satisfies the required properties. The measure μ is invariant if $\pi \sum_{a \in A} \nu(a) = \pi$. Note that rational measures with dimension 1 are the Bernoulli measures.

We give below an example of a rational measure. Consider the stochastic representation $\langle \pi, \nu, \rho \rangle$ of dimension 2 where π is the row vector $(1, 0)$, ρ is the column vector $\mathbf{1}$ and the morphism ν from $\{0, 1\}^*$ into 2×2 -matrices is defined by

$$\nu(0) = \frac{1}{3} \begin{pmatrix} 1 & 1 \\ 0 & 3 \end{pmatrix} \quad \text{and} \quad \nu(1) = \frac{1}{3} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

An equivalent weighted automaton is pictured in Figure 4. The measure defined by this representation is given for each word w of length n ending with a block of $0 \leq k \leq n$ zeros by $\mu(w) = (1 + (1 + 3 + \dots + 3^{k-1}))/3^n = (3^k + 1)/23^n$. We will see that this measure does not meet our expectations because it is not irreducible: the weighted automaton pictured in Figure 4 is not strongly connected.

2.2 Automata and ambiguity

We refer the reader to [19] for a complete introduction to automata accepting (infinite) sequences of symbols. A (*Büchi*) *automaton* \mathcal{A} is a tuple $\langle Q, A, \Delta, I, F \rangle$ where Q is the finite state set, A the alphabet, $\Delta \subseteq Q \times A \times Q$ the transition relation, $I \subseteq Q$ the set of initial states and F is the set of final states. A transition is a tuple $\langle p, a, q \rangle$ in $Q \times A \times Q$ and it is written $p \xrightarrow{a} q$. A *finite run* in \mathcal{A} is a finite sequence of consecutive transitions,

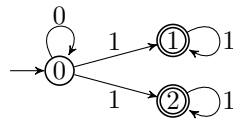
$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

Its *label* is the word $a_1 a_2 \cdots a_n$. An *infinite run* in \mathcal{A} is a sequence of consecutive transitions,

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \cdots$$

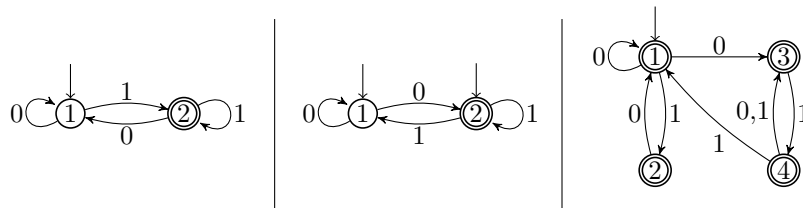
A run is *initial* if its first state q_0 is initial, that is, belongs to I . A run is called *final* if it visits infinitely often a final state. An infinite run is *accepting* if it is both initial and final. A sequence is *accepted* if it is the label of an accepting run. The set of accepted sequences is said to be *accepted* by the automaton. As usual, an automaton is *deterministic* if it has only one initial state, that is $\#I = 1$ and if $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ are two of its transitions with the same starting state and the same label, then $q = q'$. The automaton pictured in Figure 1 accepts the set $0^*1^{\mathbb{N}}$ of sequences having some 0s and then only 1s. The leftmost automaton pictured in Figure 2 is deterministic while the middle one is not. Both accept the set of sequences having infinitely many 1s. An automaton is *trim* if each state occurs in an accepting run.

For each state q , its *future* (respectively *bi-future*) is the set $F(q)$ (respectively, $F_2(q)$) of sequences labelling a final run (respectively, at least two final runs) starting from q . Let $\bar{F}(q)$ (respectively, $\bar{F}_2(q)$) be the set of sequences labelling at least one (respectively, two) infinite run starting from q which might be final or not. Note that if the automaton is trim, $\bar{F}(q)$ is indeed the topological closure of $F(q)$ but that $\bar{F}_2(q)$ might not be the topological closure of $F_2(q)$ as shown by the automaton pictured in Figure 1. The *past* $P(q)$ of a state q is the set of finite words labelling a run ending in q . For an automaton \mathcal{A} , we let $\text{fact}(\mathcal{A})$ denote the set of finite words labelling some run in \mathcal{A} . Therefore $\text{fact}(\mathcal{A}) = \bigcup_{q \in Q} P(q)$ where Q is the state set of \mathcal{A} .



■ **Figure 1** $\bar{F}_2(0) = 0^*1^{\mathbb{N}}$ and $\overline{\bar{F}_2(0)} = 0^*1^{\mathbb{N}} \cup \{0^{\mathbb{N}}\}$.

An automaton is *unambiguous* (for finite words) if for each states $p, q \in Q$ and each word w , there is at most one run $p \xrightarrow{w} q$ from p to q labelled by w . Each automaton which is either deterministic or reverse-deterministic is unambiguous.



■ **Figure 2** Three unambiguous automata.

The three automata pictured in Figure 2 are unambiguous. The leftmost one is deterministic and $F(1) = F(2) = (0^*1)^{\mathbb{N}}$, $\bar{F}(1) = \bar{F}(2) = \{0, 1\}^{\mathbb{N}}$ and $F_2(1) = F_2(2) = \emptyset$. The middle one is reverse deterministic (that is, becomes deterministic if transitions are reversed), $F(1) = 0(0^*1)^{\mathbb{N}}$, $F(2) = 1(0^*1)^{\mathbb{N}}$, $\bar{F}(1) = 0\{0, 1\}^{\mathbb{N}}$, $\bar{F}(2) = 1\{0, 1\}^{\mathbb{N}}$ and $F_2(1) = F_2(2) = \emptyset$. The rightmost one is neither deterministic nor reverse deterministic but it is unambiguous. Note however that $F_2(1)$ is not empty: $F_2(1) \supset 0^*(01)^{\mathbb{N}}$. An ambiguous automaton is pictured in Figure 3 below.

With each stochastic representation $\langle \pi, \nu, \mathbf{1} \rangle$ of a rational measure is associated an automaton whose state set is $P = \{1, \dots, m\}$ where m is the common dimension of all matrices $\nu(a)$. For each states $p, q \in P$, there is a transition $p \xrightarrow{a} q$ whenever $\nu(a)_{p,q} > 0$. The initial states are those states q in P such that $\pi_q > 0$. Due to this automaton, $\bar{F}(p)$ is well-defined for a state $p \in P$. The representation is called *irreducible* if this automaton is strongly connected. A rational measure is called *irreducible* if it has at least one irreducible representation.

A strongly connected component C of graph (respectively automaton) is called *terminal* if it cannot be left, that is, if $p \rightarrow q$ is a edge with $p \in C$, then $q \in C$.

3 Main result

Ambiguity of automata has been defined using finite words: an automaton is ambiguous if some finite word w is the label of two different runs from a state p to a state q . If the automaton is trim, this implies that some sequence of the form wy is the label of two different runs from p . The converse of this implication does not hold in general. The third automaton pictured in Figure 2 is unambiguous although the sequence $(01)^{\mathbb{N}} = 0101\dots$ is the label of the following two accepting runs starting from state 1.

$$\begin{aligned}
 &1 \xrightarrow{0} 1 \xrightarrow{1} 2 \xrightarrow{0} 1 \xrightarrow{1} 2 \xrightarrow{0} 1 \xrightarrow{1} \dots \\
 &1 \xrightarrow{0} 3 \xrightarrow{1} 4 \xrightarrow{0} 3 \xrightarrow{1} 4 \xrightarrow{0} 3 \xrightarrow{1} \dots
 \end{aligned}$$

34:6 Ambiguity Through the Lens of Measure Theory

However, the set $F_2(1)$ is contained in $(0+1)^*(01)^{\mathbb{N}}$ and it is thus countable and of measure 0 for the uniform measure. Note that if each transition $p \xrightarrow{1} q$ is replaced by the two transitions $p \xrightarrow{1} q$ and $p \xrightarrow{2} q$, the set $F_2(1)$ is not anymore countable but it is still of measure 0 as a subset of $\{0, 1, 2\}^{\mathbb{N}}$.

The following theorem provides a characterization of ambiguity using measure theory. More precisely, it states that a strongly connected automaton is unambiguous whenever the measure of sequences labelling two runs is negligible, that is, of measure zero.

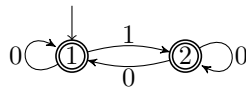
► **Theorem 1.** *Let \mathcal{A} be a strongly connected Büchi automaton and let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(\mathcal{A})$. The following conditions are equivalent.*

- i) *The automaton \mathcal{A} is unambiguous.*
- ii) *For each state q of \mathcal{A} , $\mu(\bar{F}_2(q)) = 0$.*
- iii) *There is a state q of \mathcal{A} such that $\mu(F_2(q)) = 0$.*

The equality $\text{supp}(\mu) = \text{fact}(\mathcal{A})$ is called the *full-support condition* in [7] because the inclusion $\text{supp}(\mu) \subseteq \text{fact}(\mathcal{A})$ is implicit in [7]. The irreducibility of the measure ensures that it does not put too much weight on too small sets (See example after Proposition 2). The measure used to quantify this ambiguity must also be compatible with the automaton. More precisely, its support must be equal to the set of finite words labelling at least one run in the automaton. If this condition is not fulfilled, the result may not hold as it is shown by the following two examples below.

We first explain, given a strongly connected automaton \mathcal{A} , how to construct an irreducible measure μ such that equality $\text{supp}(\mu) = \text{fact}(\mathcal{A})$ holds. Informally, the construction consists in assigning positive weights to states and transitions of \mathcal{A} such that the following two conditions are satisfied. The weights of the states must sum up to 1 (this is a distribution) and for each state q , the weights of the transitions outgoing from q must sum up to 1. Assigning weights to states is the same as defining a stochastic Q -vector π with positive entries. Assigning weights to transitions is the same as defining a $Q \times Q$ -matrix $\nu(a)$ for each symbol $a \in A$ such that the matrix $\sum_{a \in A} \nu(a)$ is stochastic and such that for each states p, q and each symbol a , the (p, q) entry of $\nu(a)$ is positive if and only if $p \xrightarrow{a} q$ is a transition of \mathcal{A} . The measure given by the representation $\langle \pi, \nu, \mathbb{1} \rangle$ is then a rational and irreducible measure such that $\text{supp}(\mu) = \text{fact}(\mathcal{A})$. It is irreducible because the automaton \mathcal{A} is strongly connected and each transition gets a positive weight. It satisfies $\text{supp}(\mu) = \text{fact}(\mathcal{A})$ because weights of states and transition are all positive. Note that not all compatible measures can be obtained that way.

Consider again the third automaton pictured in Figure 2. Let μ be the probability measure putting weight $1/2$ on each of the sequences $(01)^{\mathbb{N}}$ and $(10)^{\mathbb{N}}$ and zero everywhere else. More formally, it is defined $\mu((01)^{\mathbb{N}}) = \mu((10)^{\mathbb{N}}) = 1/2$ and $\mu(\{0, 1\}^{\mathbb{N}} \setminus \{(01)^{\mathbb{N}}, (10)^{\mathbb{N}}\}) = 0$. The measure $\mu(F_2(1)) = 1/2$ is non-zero although the automaton is unambiguous because the support $(01)^* + (10)^*$ of this measure μ is strictly contained in the set of words labelling a run in this automaton. This latter set is actually the set $\{0, 1\}^*$ of all finite words over $\{0, 1\}$.



■ **Figure 3** An ambiguous automaton accepting $(0+10)^{\mathbb{N}}$.

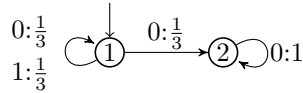
Consider the automaton pictured in Figure 3. It accepts the set X of sequences with no consecutive 1s. It is ambiguous because the word 00 is the label of the two runs $2 \xrightarrow{0} 1 \xrightarrow{0} 1$ and $2 \xrightarrow{0} 2 \xrightarrow{0} 1$. The uniform measure $\mu(X)$ is zero. Therefore, both numbers $\mu(F_2(1))$ and $\mu(F_2(2))$ are zero although the automaton is ambiguous. This comes from the fact that the support $\{0, 1\}^*$ of the uniform measure strictly contains the set $\text{fact}(X)$. This latter set is the set $(0 + 10)^*(1 + \varepsilon)$ of finite words with no consecutive 1s.

4 Reduction to closed sets

The purpose of this section is to show that the measure $\mu(X)$ of a rational set of sequences is closely related to the measure $\mu(\overline{X})$ of its closure as long as the measure μ is compatible with X . The main result of this section is the following proposition which is used in the proof of Theorem 1. The rest of the section is devoted to the proof of the proposition.

► **Proposition 2.** *Let \mathcal{A} be a strongly connected Büchi automaton, q be a state of \mathcal{A} and w be a word in $P(q)$. Let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(\mathcal{A})$. Then $\mu(wF(q)) = \mu(w\overline{F}(q))$.*

The following example shows that the irreducibility assumption of the measure is indeed necessary. Consider the measure given by the weighted automaton pictured in Figure 4.



■ **Figure 4** A weighted automaton defining a non-irreducible measure.

It realizes the measure already considered at the end of Section 2.1. This measure μ is equivalently defined by $\mu(w) = (1, 0)\nu(w)\mathbf{1}$ for each finite word w where the morphism ν from $\{0, 1\}^*$ into 2×2 -matrices is given by

$$\nu(0) = \frac{1}{3} \begin{pmatrix} 1 & 1 \\ 0 & 3 \end{pmatrix} \quad \text{and} \quad \nu(1) = \frac{1}{3} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

The weight of a word $w1$ ending with a 1 is $3^{-|w|-1}$. Therefore, the sum of these weights when w ranges over all words of length k over the alphabet $\{0, 1\}$ is given by

$$\sum_{|w|=k} \mu(w1) = \frac{2^k}{3^{k+1}} \quad \text{and} \quad \sum_{|w| \geq n} \mu(w1) = \sum_{k \geq n} \frac{2^k}{3^{k+1}} = \frac{2^n}{3^n}$$

Let $X = (0^*1)^\mathbb{N}$ be the set of sequences having infinitely many occurrences of the symbol 1. This set is equal to $F(1)$ in the leftmost automaton pictured in Figure 2. Since X is contained in the union $\bigcup_{|w| \geq n} w1\{0, 1\}^\mathbb{N}$ for each integer $n \geq 0$, the measure of X satisfies $\mu(X) \leq 2^n/3^n$ for each integer $n \geq 0$. This proves that the measure of X is 0 although the measure of its closure $\overline{X} = \{0, 1\}^\mathbb{N}$ is 1.

If \mathcal{A} is an automaton and $P \subseteq Q$ is a subset of its state set Q , we let $P \cdot w$ denote the subset $P' \subseteq Q$ defined by $P' = \{q : \exists p \in P \ p \xrightarrow{w} q\}$. If P is a singleton set $\{q\}$, we write $q \cdot w$ for $\{q\} \cdot w$. By a slight abuse of notation, we also write $q \cdot w = p$ for $\{q\} \cdot w = \{p\}$. If \mathcal{A} is deterministic, $q \cdot w$ is either the empty set or a singleton set.

The following easy lemma states that, in each strongly connected automaton, there exists a run using all transitions.

34:8 Ambiguity Through the Lens of Measure Theory

► **Lemma 3.** *Let \mathcal{A} be a strongly connected deterministic automaton. There exists a finite word w such that:*

- i) *there exists a state q such that $q \cdot w$ is non-empty,*
- ii) *for each state q of \mathcal{A} , if $q \cdot w$ is non-empty, each transition of \mathcal{A} occurs in the run $q \xrightarrow{w} q \cdot w$.*

Let $\langle \pi, \nu, \mathbb{1} \rangle$ be the representation of a rational measure μ . Its support $\text{supp}(\nu)$ is defined by $\text{supp}(\nu) = \{w : \nu(w) \neq 0\}$. It obviously satisfies $\text{supp}(\mu) \subseteq \text{supp}(\nu)$. This inclusion can be strict as shown by the following example but it becomes an equality as soon as $\text{supp}(\mu)$ is *factorial*, that is closed under taking factor.

Let μ be the measure defined by $\mu(0w) = 0$ and $\mu(1w) = 2^{-|w|}$ for each word w in $\{0, 1\}^*$. It is rational because it is defined by $\mu(w) = (0, 1)\nu(w)\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ where the morphism ν from $\{0, 1\}^*$ into 2×2 -matrices is given by $\nu(0) = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$ and $\nu(1) = \frac{1}{2} \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$. The support of this measure μ is $\text{supp}(\mu) = 1\{0, 1\}^*$ but the support of the morphism ν is $\text{supp}(\nu) = \{w \in \{0, 1\}^* : \nu(w) \neq 0\}$ is $\{0, 1\}^*$. The support of a rational measure and the support of one of its representations might not coincide in general but they do coincide as soon as $\text{supp}(\mu)$ is factorial as stated by the following lemma.

► **Lemma 4.** *Let μ be an irreducible rational measure and let $\langle \pi, \nu, \mathbb{1} \rangle$ be an irreducible representation of μ . Then $\text{supp}(\mu)$ is factorial if and only if the equality $\text{supp}(\mu) = \text{supp}(\nu)$ holds.*

In the rest of the paper, the support of each measure is a factorial set and both supports coincide.

The following lemma states the of sequences having finitely many occurrence of some finite word has zero measure. The result is well-known when the measure is a Markov measure and the word is just a symbol [8, Thm 3.3]. The irreducibility and rationality of the measure are both crucial.

► **Lemma 5.** *Let μ be an irreducible rational measure such that $\text{supp}(\mu)$ is factorial and let w be a word in $\text{supp}(\mu)$. Then*

$$\mu(\{x : |x|_w < \infty\}) = 0.$$

where $|x|_w$ is the number of occurrences of w in x .

We let $\xrightarrow{*}$ denote the accessibility relation in an automaton. We write $p \xrightarrow{*} q$ if there is a run from p to q . If P and P' are two subsets of states of an automaton, we write $P \xrightarrow{*} P'$ whenever there is a run from a state in P to a state in P' . This relation is not transitive in general but it is when each considered subset is contained in a strongly connected component. The following lemma gives a property of Muller automata accepting the same set as a strongly connected Büchi automaton.

► **Lemma 6.** *Let $X \subseteq A^{\mathbb{N}}$ be a non-empty set of sequences accepted by a strongly connected Büchi automaton and let $w \in A^*$ be a finite word. Let \mathcal{A} be a Muller automaton accepting the set wX and let \mathcal{T} be its table of accepting subsets of states. Let F be an element of \mathcal{T} such that $F' \in \mathcal{T}$ and $F \xrightarrow{*} F'$ imply $F' \xrightarrow{*} F$. Then the strongly connected component containing F also belongs to the table \mathcal{T} .*

A subset P of states of a Muller automaton \mathcal{A} is called *essential* if there is an infinite run in \mathcal{A} such that P is the set of states that occur infinitely often along this run.

Proof of Proposition 2. The proof of the proposition is reduced to proving that $\mu(w\bar{F}(q) \setminus wF(q)) = 0$. We consider a Muller automaton accepting $wF(q)$ with a table \mathcal{T} . Note that a Muller automaton accepting $w\bar{F}(q)$ is obtained by replacing the table \mathcal{T} by the table $\bar{\mathcal{T}}$ which contains each essential set of states which can access an essential set of states in \mathcal{T} . The difference set $w\bar{F}(q) \setminus wF(q)$ is thus accepted by the same Muller automaton with the table $\mathcal{T} \setminus \bar{\mathcal{T}}$. By Lemma 6, each maximal essential state is in the table \mathcal{T} . By combining Lemmas 3 and 5, it is clear that $\mu(w\bar{F}(q) \setminus wF(q)) = 0$. ◀

5 Proof for closed sets

Thanks to Proposition 2, it is sufficient to study closed sets. As the initial states of the automaton are not relevant for the statement of Theorem 1, we consider automata where all states are initial and final, that is, $I = F = Q$. It turns out that these automata accept shift spaces that we now introduce.

The *shift map* is the function σ which maps each sequence $(x_i)_{i \geq 1}$ to the sequence $(x_{i+1})_{i \geq 1}$ obtained by removing its first element. A *shift space* is a subset X of $A^{\mathbb{N}}$ which is closed for the usual product topology and such that $\sigma(X) = X$. A classical example of a shift space is the *golden mean shift*: it is the set $\{0, 10\}^{\mathbb{N}}$ of sequences with no consecutive 1s. We refer the reader to [18] for a complete introduction to shift spaces.

If a shift space is accepted by some trim Büchi automaton, it is also accepted by the same automaton in which each state is made initial and final. A shift space is called *sofic* if it is accepted by some automaton. A sofic shift is called *irreducible* if it is accepted by a strongly connected Büchi automaton. It is well-known [18] that each shift space is characterized by the set of factors of its sequences. Let us recall that $\text{fact}(X)$ denotes the set of factors of a shift space X .

There is a unique, up to isomorphism, deterministic automaton accepting an irreducible sofic shift with the minimal number of states [18, Thm 3.3.18]. This *minimal automaton* is also referred to as either its Shannon cover or its Fischer cover. It can be obtained from any automaton accepting the shift space via determinizing and state-minimizing algorithms, e.g., [18, pp. 92], [16, pp. 68]. The minimal automaton of the golden mean shift is the leftmost automaton pictured in Figure 5. A *synchronizing word* of a strongly connected automaton is a word w such that there is a unique state q such that $w \in P(q)$. The word 1 is a synchronizing word of both automata pictured in Figure 5. The minimal automaton of a sofic shift has always at least one synchronizing word [18, Prop. 3.3.16].



■ **Figure 5** Two automata accepting the golden mean shift.

The next lemma states in particular (for $w = \varepsilon$) that the future of each state of the minimal automaton of a sofic shift has a positive measure. The

► **Lemma 7.** *Let r be a state of the minimal automaton of an irreducible sofic shift space X and let w be a synchronizing word such that $w \in P(r)$. Let μ be a rational measure such that $\text{supp}(\mu) = \text{fact}(X)$. Then $\mu(w\bar{F}(r)) = \mu(wA^{\mathbb{N}}) > 0$.*

The following lemma establishes a link between any automaton accepting a sofic system and its minimal automaton. It allows us to transfer the result of the previous lemma to non-minimal automata.

► **Lemma 8.** *Let \mathcal{A} be a strongly connected automaton accepting an irreducible sofic shift space X . Let w be a word and q be a state of \mathcal{A} such that $w \in P(q)$. There exists a state r of the minimal automaton of X and a synchronizing word v of this minimal automaton such that $wv \in P(r)$ and $\bar{F}(q) \cap vA^{\mathbb{N}} = v\bar{F}(r)$.*

Combining Lemmas 7 and 8 yields the following result.

► **Lemma 9.** *Let \mathcal{A} be an strongly connected automaton accepting an irreducible sofic shift X . Let q be a state of \mathcal{A} and let w be a word such that $w \in P(q)$. Let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(X)$. Then $\mu(w\bar{F}(q)) > 0$.*

Proof. By Lemma 8, there exists a state r of the minimal automaton of X and a synchronizing word v such that $wv \in P(r)$ and $\bar{F}(q) \cap vA^{\mathbb{N}} = v\bar{F}(r)$. This implies that $w\bar{F}(q) \cap wvA^{\mathbb{N}} = wv\bar{F}(r)$. By Lemma 7, the measure $\mu(wv\bar{F}(r))$ is positive and thus $\mu(w\bar{F}(q)) > 0$. ◀

It is a very classical result that not all regular sets of sequences are accepted by deterministic Büchi automata. This is the reason why Muller automata with a more involved acceptance condition were introduced. Landweber’s theorem states that a regular set of sequences is accepted by a deterministic Büchi automaton if and only if it is a G_δ -set (that is Π_0^2) [19, Thm I.9.9]. This implies in particular that regular and closed sets¹ are accepted by deterministic Büchi automata. Regular and closed sets are actually accepted by deterministic Büchi automata in which each state is final [19, Prop III.3.7].

Lemma 9 states that the future of a state in automaton with all states final has a positive measure. The following provides a converse. It states that a closed set F with positive measure contains the future of a state of the minimal automaton, prefixed by some word w . The prefix w is really needed because the closed set F can be arbitrarily small.

► **Lemma 10.** *Let X be a sofic shift space and let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(X)$. Let F be a regular and closed set contained in X . If $\mu(F) > 0$, there exists a word w and a state r of the minimal automaton of X such that $w \in P(r)$ and $w\bar{F}(r) \subseteq F$.*

Before proceeding to the proof of the lemma, we show that even in the case of the full shift, that is $X = A^{\mathbb{N}}$, both hypothesis of being regular and closed are necessary. Since the minimal automaton of the full shift has a single state r satisfying $\bar{F}(r) = A^{\mathbb{N}}$, the lemma can be, in that case, rephrased as follows. If $\mu(F) > 0$ where μ is the uniform measure, then there exists a word w such that $wA^{\mathbb{N}} \subseteq F$.

Being regular is of course not sufficient because the set $(0^*1)^{\mathbb{N}}$ of sequences having infinitely many occurrences of 1 is regular and has measure 1 but does not contain any set of the form $wA^{\mathbb{N}}$. Being closed is also not sufficient as it is shown by the following example. Let X be the set of sequences such that none of their non-empty prefixes of even length is a palindrome. The complement of X is equal to the following union

$$\bigcup_{n \geq 1} Z_n \quad \text{where} \quad Z_n = \bigcup_{|w|=n} w\tilde{w}A^{\mathbb{N}}$$

and where \tilde{w} stands for the reverse of w . Suppose for instance that the alphabet is $A = \{0, 1\}$. The measure of Z_n is equal 2^{-n} because there are 2^n words of length n and the measure of each cylinder $w\tilde{w}A^{\mathbb{N}}$ is 2^{-2n} . Furthermore, the set $Z_1 \cup Z_2$ is equal to

¹ Not to be confused with *regular closed sets* which are equal to the closure of their interior [13, Chap. 4].

$00A^{\mathbb{N}} \cup 11A^{\mathbb{N}} \cup 0110A^{\mathbb{N}} \cup 1001A^{\mathbb{N}}$ whose measure is $5/8$. This shows that the measure of the complement of X is bounded by $5/8 + \sum_{n \geq 3} 2^{-n} = 7/8$ (Note that this is really an upper bound as the sets Z_n are not pairwise disjoint). Therefore X has a positive measure but it does not contain any cylinder. Indeed, in each cylinder $wA^{\mathbb{N}}$, the cylinder $w\tilde{w}A^{\mathbb{N}}$ is out of X .

The following result is trivially true when the measure μ is shift invariant because $\mu(F) = \sum_{|w|=m} \mu(wF)$ but it does not hold in general.

► **Lemma 11.** *Let X be a sofic shift space and let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(X)$. Let F be a regular and closed set contained in X . If $\mu(F) = 0$, then $\mu(wF) = 0$ for each finite word w .*

Proof. We prove that $\mu(wF) > 0$ implies $\mu(F) > 0$. Suppose that $\mu(wF) > 0$. Since wF is also regular and closed, there exists, by Lemma 10, a word u and a state r of the minimal automaton of X such that $u \in P(r)$ and $u\bar{F}(r) \subseteq wF$. This latter inclusion implies that either u is a prefix of w or w is a prefix of u . In the first case, that is $w = uv$ for some word v , the inclusion is equivalent to $\bar{F}(r) \subseteq vF$. Let s be state such that $r \xrightarrow{v} s$. Then $v\bar{F}(s)$ is contained in $\bar{F}(r)$ and thus $\bar{F}(s) \subseteq F$. By Lemma 9, $\mu(\bar{F}(s)) > 0$ and thus $\mu(F) > 0$. In the second case, that is $u = wv$, for some v , the inclusion is equivalent to $v\bar{F}(r) \subseteq F$. Again by Lemma 9, $\mu(v\bar{F}(r)) > 0$ and thus $\mu(F) > 0$. ◀

The following lemma is an extension to pairs of futures of states of the result of Lemma 7 for one state.

► **Lemma 12.** *Let \mathcal{A} be strongly connected automaton accepting a shift space X . Let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(X)$. Let q and q' two states of \mathcal{A} such that $\mu(\bar{F}(q) \cap \bar{F}(q')) > 0$. Then there exists a word w such that $\bar{F}(q) \cap wA^{\mathbb{N}} = \bar{F}(q') \cap wA^{\mathbb{N}}$.*

Proof. We claim that there exists a word w and a state r of the minimal automaton of X such that $w \in P(r)$ and

$$\bar{F}(q) \cap wA^{\mathbb{N}} = \bar{F}(q') \cap wA^{\mathbb{N}} = w\bar{F}(r).$$

Let F be the closed set $\bar{F}(q) \cap \bar{F}(q')$. By Lemma 10 applied to F , there exists a word u and a state s of the minimal automaton of X such that $u \in P(s)$ and

$$\begin{aligned} u\bar{F}(s) &\subseteq \bar{F}(q) \\ u\bar{F}(s) &\subseteq \bar{F}(q') \end{aligned}$$

Let v be a synchronizing word of the minimal automaton of X such that $s \cdot v$ is not empty. Let w be the word uv and let r be the state $s \cdot v$. Since $u \in P(s)$ and $r = s \cdot v$, $w \in P(r)$. We claim that $\bar{F}(q) \cap wA^{\mathbb{N}} = w\bar{F}(r)$. Suppose first that x belongs to $\bar{F}(q) \cap wA^{\mathbb{N}}$. The sequence x is then equal to wx' for some sequence x' and it is the label of a run in the minimal automaton of X . Since $w = uv$ and v is synchronizing, the sequence x' must belong to $\bar{F}(r)$. Suppose conversely that x belongs to $w\bar{F}(r)$. It is then equal to uvx' for some x' in $\bar{F}(r)$. Since $r = s \cdot v$, $vx' \in \bar{F}(s)$. It follows from the inclusion $u\bar{F}(s) \subseteq \bar{F}(q)$ that x belongs to $\bar{F}(q)$. This completes the proof of the equality $\bar{F}(q) \cap wA^{\mathbb{N}} = w\bar{F}(r)$. By symmetry, the equality $\bar{F}(q') \cap wA^{\mathbb{N}} = w\bar{F}(r)$ also holds and the proof is completed. ◀

This last lemma establishes a link between unambiguity of an automaton and measures of futures of its states. More precisely, it states that the future of two states that can be reached from the same state and reading the same word have an intersection of zero measure. Its proof is more combinatorial than previous ones.

34:12 Ambiguity Through the Lens of Measure Theory

► **Lemma 13.** *Let \mathcal{A} be an unambiguous strongly connected automaton accepting a shift space X . Let μ be an irreducible rational measure such that $\text{supp}(\mu) = \text{fact}(X)$. If there are two runs $p \xrightarrow{u} q$ and $p \xrightarrow{u} q'$, with $q \neq q'$, then $\mu(\bar{F}(q) \cap \bar{F}(q')) = 0$.*

Proof. Suppose by contradiction that $\mu(\bar{F}(q) \cap \bar{F}(q')) > 0$. There exists, by Lemma 12, a word v such that $\bar{F}(q) \cap vA^{\mathbb{N}} = \bar{F}(q') \cap vA^{\mathbb{N}}$. Let $q \cdot v$ (respectively $q' \cdot v$) be the set $\{q_1, \dots, q_r\}$ (respectively $\{q'_1, \dots, q'_{r'}\}$). Since $\bar{F}(q) \cap vA^{\mathbb{N}} = \bar{F}(q') \cap vA^{\mathbb{N}}$, the equality $\bar{F}(q_1) \cup \dots \cup \bar{F}(q_r) = \bar{F}(q'_1) \cup \dots \cup \bar{F}(q'_{r'})$ holds. Since the automaton is strongly connected, there is a run $q_1 \xrightarrow{w} p$ from q_1 to p . Combining this run with the run $p \xrightarrow{u} q \xrightarrow{v} q_1$ yields the cyclic run $q_1 \xrightarrow{wuv} q_1$. Since $\bar{F}(q_1) \cup \dots \cup \bar{F}(q_r) = \bar{F}(q'_1) \cup \dots \cup \bar{F}(q'_{r'})$, the sequence $(wuv)^{\mathbb{N}} = wuvwuv \dots$ belongs to a set $\bar{F}(q'_i)$ for some $1 \leq i \leq r'$. By symmetry, it can be assumed that $(wuv)^{\mathbb{N}} \in \bar{F}(q'_1)$. There exists then a run starting from q'_1 with label $(wuv)^{\mathbb{N}}$. This run can be decomposed

$$q'_1 \xrightarrow{wuv} p_1 \xrightarrow{wuv} p_2 \xrightarrow{wuv} p_3 \dots$$

Since there are finitely many states, there are two integers $k, \ell \geq 1$ such that $p_k = p_{k+\ell}$. There are then the following two runs from p to $p_k = p_{k+\ell}$ with the same label $(uvw)^{k+\ell}uv$.

$$\begin{aligned} p &\xrightarrow{u} q \xrightarrow{v} q_1 \xrightarrow{(wuv)^{k-1}} q_1 \xrightarrow{w} p \xrightarrow{uv} q'_1 \xrightarrow{(wuv)^\ell} p_k \\ p &\xrightarrow{u} q' \xrightarrow{v} q'_1 \xrightarrow{(wuv)^{k+\ell}} p_{k+\ell} \end{aligned}$$

This is a contradiction with the fact that \mathcal{A} is unambiguous. ◀

Proof of Theorem 1. We first prove that (i) implies (ii). Suppose that the automaton \mathcal{A} is unambiguous. We show that $\mu(\bar{F}_2(p)) = 0$ for each state p . We start by a decomposition of the set $\bar{F}_2(p)$. Let $x = a_1a_2a_3 \dots$ be a sequence in $\bar{F}_2(p)$ and let ρ and ρ' be the two different runs labelled by x . Suppose that

$$\begin{aligned} \rho &= q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \dots \\ \rho' &= q'_0 \xrightarrow{a_1} q'_1 \xrightarrow{a_2} q'_2 \xrightarrow{a_3} q'_3 \dots \end{aligned}$$

where $q_0 = q'_0 = p$. Let n be the least integer such that $q_n \neq q'_n$. Let a be the symbol a_n , w be the prefix $a_1 \dots a_{n-1}$ and x' be the tail $a_{n+1}a_{n+2}a_{n+3} \dots$. The sequence x is equal to wax' and there is a finite run $q_0 \xrightarrow{w} q_{n-1}$, two transitions $q_{n-1} \xrightarrow{a} q_n$ and $q_{n-1} \xrightarrow{a} q'_n$, and the tail x' belongs to the intersection $\bar{F}(q_n) \cap \bar{F}(q'_n)$. We have actually proved the following equality expressing $\bar{F}_2(p)$ in term of a union of intersections of sets $\bar{F}(q)$.

$$\bar{F}_2(p) = \bigcup_{\substack{p \xrightarrow{w} p' \\ p' \xrightarrow{a} q \\ p' \xrightarrow{a} q'}} wa(\bar{F}(q) \cap \bar{F}(q'))$$

Since the union is countable, it suffices to prove that if there are two transitions $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ with $q \neq q'$, then $\mu(\bar{F}(q) \cap \bar{F}(q')) = 0$. Lemma 13 and Lemma 11 allow us to conclude.

The fact that (ii) implies (iii) is clear because the set $F_2(q)$ is contained in $\bar{F}_2(q)$ for each state q of \mathcal{A} .

We now prove that (iii) implies (i). Let q be state of \mathcal{A} and suppose that there are two different runs from state p to state r with the same label w . Let v the label of a run from q to p . This shows that $vwF(r) \subseteq F_2(q)$. Since $vw \in P(r)$, the measure $\mu(vwF(r))$ satisfies $\mu(vwF(r)) = \mu(vw\bar{F}(r))$ by Proposition 2. The measure $\mu(vw\bar{F}(r))$ satisfies $\mu(vw\bar{F}(r)) > 0$ by Lemma 10 and thus $\mu(F_2(q)) > 0$. This completes the proof of this implication. ◀

Conclusion

As a conclusion, we would like to emphasize the difficulty of extending the result to non strongly connected automata. Consider the two automata pictured in Figure 6. The leftmost one is unambiguous whereas the rightmost one is obviously ambiguous for finite words. However, $F_2(0) = \overline{F}_2(0) = 0^*1^{\mathbb{N}}$ and $F_2(q) = \emptyset$ hold for $q \neq 0$ in both automata. The only way to distinguish one automaton from the other one is to have two different measures. In order to have $\mu_1(F_2(0)) = 0$ for the leftmost automaton, the measure μ_1 should put all the weight on $0^{\mathbb{N}}$: $\mu_1(0^{\mathbb{N}}) = 1$. In order to have $\mu_2(F_2(0)) > 0$ for the rightmost automaton, the measure μ_2 should put some weight on a sequence $0^n 1^{\mathbb{N}}$ for some integer $n \geq 0$. It is not clear why the measures should be different because the set 0^*1^* of finite words labelling some run is the same in both automata.



■ **Figure 6** Two non strongly connected automata.

References

- 1 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- 2 V. Becher and O. Carton. Normal numbers and computer science. In V. Berthé and M. Rigo, editors, *Sequences, Groups, and Number Theory*, Trends in Mathematics Series, pages 233–269. Birkhäuser, 2018.
- 3 J. Berstel and D. Perrin. *Theory of Codes*. Academic Press, 1984.
- 4 J. Berstel and Ch. Reutenauer. *Noncommutative Rational Series with Applications*. Cambridge University Press, 2010.
- 5 É. Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti Circ. Mat. Palermo*, 27:247–271, 1909.
- 6 N. Bousquet and Ch. Löding. Equivalence and inclusion problem for strongly unambiguous Büchi automata. In *LATA 2010*, volume 6031 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2010. doi:10.1007/978-3-642-13089-2_10.
- 7 M. Boyle and K. Petersen. *Entropy of Hidden Markov Processes and Connections to Dynamical Systems*, volume 385 of *London Mathematical Society Lecture Notes*, chapter Hidden Markov processes in the context of symbolic dynamics, pages 5–71. Cambridge University Press, 2011.
- 8 P. Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, 2008.
- 9 O. Carton. Preservation of normality by unambiguous transducers. *CoRR*, abs/2006.00891, 2022. arXiv:2006.00891.
- 10 O. Carton and M. Michel. Unambiguous Büchi automata. *Theoret. Comput. Sci.*, 297:37–81, 2003.
- 11 Ch. Choffrut and S. Grigorieff. Uniformization of rational relations. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 59–71. Springer, 1999.
- 12 Th. Colcombet. Unambiguity in automata theory. In *Descriptive Complexity of Formal Systems*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015.
- 13 P. R. Halmos. *Lectures on Boolean algebras*. Von Nostrand, 1963.

- 14 G. Hansel and D. Perrin. Mesures de probabilité rationnelles. In M. Lothaire, editor, *Mots*, pages 335–357. Hermes, 1990.
- 15 M. Holzer and M. Kutrib. Descriptive complexity of (un)ambiguous finite state machines and pushdown automata. In *Reachability Problems*, pages 1–23. Springer, 2010. doi:10.1007/978-3-642-15349-5_1.
- 16 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- 17 D. Isaak and Ch. Löding. Efficient inclusion testing for simple classes of unambiguous ω -automata. *Inf. Process. Lett.*, 112(14-15):578–582, 2012.
- 18 D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.
- 19 D. Perrin and J.-É. Pin. *Infinite Words*. Elsevier, 2004.
- 20 G. Pighizzini. Two-way finite automata: Old and recent results. *Electronic Proceedings in Theoretical Computer Science*, 90:3–20, 2012. doi:10.4204/eptcs.90.1.
- 21 J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
- 22 E. Schmidt. Succinctness of descriptions of context-free, regular, and finite languages. *DAIMI Report Series*, 7(84), 1978. doi:10.7146/dpb.v7i84.6500.
- 23 R. E. Stearns and H. B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985. doi:10.1137/0214044.

Phase Semantics for Linear Logic with Least and Greatest Fixed Points

Abhishek De ✉

IRIF, Université Paris Cité, CNRS & INRIA $\pi.r^2$, France

Farzad Jafarrahmani ✉

IRIF, Université Paris Cité, CNRS & INRIA $\pi.r^2$, France

Alexis Saurin ✉

IRIF, CNRS, Université Paris Cité & INRIA $\pi.r^2$, France

Abstract

The truth semantics of linear logic (*i.e.* phase semantics) is often overlooked despite having a wide range of applications and deep connections with several denotational semantics. In phase semantics, one is concerned about the provability of formulas rather than the contents of their proofs (or refutations). Linear logic equipped with the least and greatest fixpoint operators (μ MALL) has been an active field of research for the past one and a half decades. Various proof systems are known *viz.* finitary and non-wellfounded, based on explicit and implicit (co)induction respectively.

In this paper, we extend the phase semantics of multiplicative additive linear logic (*a.k.a.* MALL) to μ MALL with explicit (co)induction (*i.e.* μ MALL^{ind}). We introduce a Tait-style system for μ MALL called μ MALL _{ω} where proofs are wellfounded but potentially infinitely branching. We study its phase semantics and prove that it does not have the finite model property.

2012 ACM Subject Classification Theory of computation \rightarrow Linear logic; Theory of computation \rightarrow Proof theory

Keywords and phrases Linear logic, fixed points, phase semantics, closure ordinals, cut elimination

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.35

Funding This research has been partially supported by ANR project *RECIPROG*, project reference ANR-21-CE48-019-01.

Abhishek De: This author has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754362.

Acknowledgements We would like to thank anonymous reviewers for their valuable comments that enhanced the clarity and presentation of this paper. We would like to thank Amina Doumane, Graham Leigh and Rémi Nollet for helpful discussions in the early phase of this work.

1 Introduction

Fixpoint logics: from truth to proofs. Fixpoint logics were first introduced in the study of inductive definability [1] in recursion theory which predates its first application in computer science as an expressive database query language [3]. In order to define the language of a fixpoint logic, one introduces explicit fixpoint construct(s) and closes it under these construct(s) thus obtaining a richer language. First order logic extended with various fixpoint operators have been extensively explored in model theory [38]. In the propositional case, the (multi)modal μ -calculus (the extension of basic modal logic K with least and greatest fixpoint operators) is probably the most well-studied. Introduced by Scott and Bakker in an unpublished manuscript, the logic has been historically studied in the formal methods and verification community [16]. More recently, there has been a growing interest in its structural proof-theory. The most important result in this direction is the completeness of Hilbert-style axiomatisations for the logic, which has turned out to be notoriously difficult [34, 51, 50].



© Abhishek De, Farzad Jafarrahmani, and Alexis Saurin;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 35; pp. 35:1–35:23



Leibniz International Proceedings in Informatics

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

Various proof systems for fixpoint logics. The setting of our paper is μ MALL, the extension of multiplicative additive linear logic by fixpoints *viz.* the propositional fragment of the logic introduced by Baelde and Miller in [7] who proposed the sequent calculus system μ MALL^{ind} for the logic. All these formalisations *viz.* the Hilbert-style axiomatisations of μ -calculus and μ MALL^{ind} employ inference rules that express an *explicit* (co)induction scheme *i.e.* the induction hypothesis must be provided explicitly. However, sequent calculi with explicit (co)induction do not have the subformula property in spite of having analyticity. In fact, it is generally accepted that we do not have *true* cut elimination for any logic equipped with a theory of inductive definitions [40]. However, one can consider an alternative formalisation of inductive reasoning *viz.* *implicit* induction, which avoids the need for explicitly specifying (co)induction invariants. This formalism generally recovers true cut elimination but at the price of infinitary axiomatisation of the fixpoints. There are two approaches to this.

The first approach is to consider a Tait-style system *i.e.* infinitary wellfounded derivations which use a so-called ω -rule with infinitely many premises of finite approximations of a fixpoint. Such rules arise in various areas of logic, notably as Carnap's rule [15] in arithmetic. A complete Tait-style system has been proposed for fixpoint logics *viz.* for the μ -calculus [35] and star-continuous action lattices [43] (where the ω -rule construes the Kleene star as an ω -iteration of finite concatenations).

The second approach is to define a non-wellfounded and/or a circular proof system with finitely branching inferences. Such systems have been extensively studied in the setting of μ MALL [44, 28, 6, 23]. Circular proofs have deep roots in the history of logic and mathematical reasoning: starting with Euclid's [27] heuristic of infinite descent through the more rigorous studies of Fermat [22]. A systematic investigation of the connection between circular proofs and reasoning by infinite descent has been carried out by Brotherston and Simpson [11, 12, 13].

Proof systems difficult to compare. Brotherston and Simpson conjectured that (in the setting of Martin-Löf's inductive definitions) circular proofs derive the same statements as finitary proofs with explicit induction. The so-called *Brotherston-Simpson conjecture* remained open for about a decade until Berardi and Tatsuta [8, 10] answered it negatively for the general case. On the other hand, if the logic contains arithmetic, the conjecture is known to be true; proved independently by Simpson [47], and Berardi and Tatsuta [9].

Note that the Brotherston-Simpson conjecture is heavily dependent on the base logic since the availability of structural rules or modal constructs induce subtle differences. For instance, the modal μ -calculus coincides on all systems. On the other hand, in Kleene Algebras, which is a substructural logic, the wellfounded, circular, and Tait-style systems are indeed different. In the setting of linear logic, in a recent work [20], the circular and the non-wellfounded system has been shown to be separate. However, the exact relation between finitary and circular system is still open. There is good reason for the difficulty. The very restricted use of structural rules in the linear setting induces a much more refined provability relation. Therefore, in this paper, we study the provability semantics *a.k.a.* the *phase semantics* of these logics as a first step of tackling the Brotherston-Simpson conjecture. Categorical semantics of circular proofs of the additive fragment have been studied in [44]. More recently, coherence space semantics have been studied for μ MALL^{ind} [24] as well as preliminary results on μ MALL _{ω} denotational semantics [26]. These semantics interpret formulas as well as their proofs thereby preserving their computational content. On the other hand, phase semantics is a coarser interpretation that allows for expressing strong invariant of linear logic provability and has been notably used to prove decidability results [36, 19] and cut admissibility results [42].

Contributions. In this paper, we shall develop a phase semantics for various proof systems for μMALL . Our first contribution is the notion of μ -phase models which are bespoke mathematical objects that are shown to be sound and complete interpretation for $\mu\text{MALL}^{\text{ind}}$. The completeness proof is intricate via Tait-Girard reducibility candidates and is inspired from [42]. Subsequently, we obtain the cut-admissibility of $\mu\text{MALL}^{\text{ind}}$ by semantic means. Furthermore, we design a new Tait-style proof system for μMALL *viz.* μMALL_ω and obtain its phase semantics and cut-admissibility. In this case, the crux of the completeness proof is the obtention of a wellfounded notion of the rank of a formula. Cut-admissibility lets us show that μMALL_ω and $\mu\text{MALL}^{\text{ind}}$ are different systems in terms of provability. Finally, we show that μMALL_ω does not have a finite model property using an idea by Lafont [36].

Organisation of the paper. The paper is organised as follows. In Section 2, we give a brief exposition of linear logic and its phase semantics, relevant fixpoint theorems and describe the syntax and relevant properties of μMALL . In Section 3, we develop the phase semantics of μMALL *wrt.* the wellfounded proof system $\mu\text{MALL}^{\text{ind}}$. In Section 4, we introduce the Tait-style proof system μMALL_ω and study its semantics, cut-admissibility and finite model property. Finally, we conclude in Section 5 discussing directions of future work. Two appendices complement the paper: a table summarizing the proof systems used in this paper is provided in Appendix A, proof details are provided in Appendix B.

Notation. Let F and G be formulas. $F(G/x)$ denotes that every occurrence of x in F is replaced by G . If x is clear from the context, we simply write it as $F(G)$. A special case is $F^n(x)$ which denotes $\overbrace{F(F(\dots(F(x))\dots))}^n$. We write a^n as a macro for $\overbrace{a\wp\dots\wp a}^n$. Let Γ be any sequent. Then, $\mathcal{L} \vdash \Gamma$ ($\mathcal{L} \vdash_{cf} \Gamma$ respectively) denotes that there is a proof (cut-free proof respectively) of Γ in the system \mathcal{L} . Finally, for any finite set S , its cardinality is $|S|$.

2 Background

2.1 Linear logic and phase semantics

Substructural logics are logics lacking at least one of the usual structural rules. In *linear logic* [30], one of the most well-studied substructural logics, sequents are effectively multisets and the use of contraction and weakening is carefully controlled. Conjunction and disjunction each have two versions in linear logic: *multiplicative* and *additive*. Consequently the units have multiplicative and additive versions as well.

	conjunction	disjunction	true	false
multiplicative	\otimes	\wp	$\mathbf{1}$	\perp
additive	$\&$	\oplus	\top	$\mathbf{0}$

The logical system thus obtained is called multiplicative-additive linear logic (MALL) and its inference rules are depicted in Figure 1 (sequents being construed as finite multisets). Full linear logic extends MALL by incorporating certain “exponential” modalities, written $?F$ and, dually, $!F$. Because of the absence of contraction and weakening, linear logic is resource-conscious *i.e.* one is concerned over the number of times that a given sentence is used in the proof of another sentence. To get a flavour of *phase semantics*, the provability semantics of linear logic [30], it is informative to characterise the set of lists Γ of formulas that make a formula F provable.

Identity rules	$\frac{}{\vdash F, F^\perp}$ (id)	$\frac{\vdash \Gamma_1, F \quad \vdash \Gamma_2, F^\perp}{\vdash \Gamma_1, \Gamma_2}$ (cut)		
Logical rules				
multiplicative connectives	$\frac{\vdash \Gamma, F_1, F_2}{\vdash \Gamma, F_1 \wp F_2}$ (\wp)	$\frac{\vdash \Gamma_1, F_1 \quad \vdash \Gamma_2, F_2}{\vdash \Gamma_1, \Gamma_2, F_1 \otimes F_2}$ (\otimes)	$\frac{}{\vdash \mathbf{1}}$ ($\mathbf{1}$)	$\frac{\vdash \Gamma}{\vdash \Gamma, \perp}$ (\perp)
additive connectives	$\frac{\vdash \Gamma, F_i}{\vdash \Gamma, F_1 \oplus F_2}$ (\oplus_i)	$\frac{\vdash \Gamma, F_1 \quad \vdash \Gamma, F_2}{\vdash \Gamma, F_1 \& F_2}$ ($\&$)	$\frac{}{\vdash \Gamma, \top}$ (\top)	No rule for $\mathbf{0}$

■ **Figure 1** Inference rules for MALL, where $i \in \{1, 2\}$.

► **Definition 1.** For a formula F , define $\text{Pr}(F) = \{\Gamma \mid \text{MALL} \vdash \Gamma, F\}$ and $\text{Pr}_{cf}(F) = \{\Gamma \mid \text{MALL} \vdash_{cf} \Gamma, F\}$.

Let us examine some properties of $\text{Pr}(F)$. First, notice that the axiom rule ensures that for any F , $F^\perp \in \text{Pr}(F)$. Invertibility of the (\perp) rule gives us $\text{Pr}(\perp)$ is the set of all provable sequents. Similar observations on the invertibility of the ($\&$) rule inform that $\text{Pr}(G \& G) = \text{Pr}(F) \cap \text{Pr}(G)$. For the (non-invertible) connectives \otimes and \oplus , we only have $\text{Pr}(F \otimes G) \supseteq \text{Pr}(F) \cdot \text{Pr}(G) = \{\Gamma, \Delta \mid \Gamma \in \text{Pr}(F), \Delta \in \text{Pr}(G)\}$ and $\text{Pr}(F \oplus G) \supseteq \text{Pr}(F) \cup \text{Pr}(G)$. This suggests that the algebraic model for linear logic should simultaneously be a monoid and lattice *i.e.* a *residuated lattice*.

$\text{Pr}(\perp)$ plays an major role in this approach, especially when considering it together with the cut inference. Indeed, for any F , one has that $\text{Pr}(F^\perp) = \{\Gamma \mid \forall \Delta \in \text{Pr}(F), \Gamma, \Delta \in \text{Pr}(\perp)\}$. This naturally suggests to consider the operation $S^\perp = \{\Gamma \mid \forall \Delta \in S, \Gamma \cdot \Delta \in \text{Pr}(\perp)\}$ which induces a closure operator $(\bullet)^{\perp\perp}$ on the set of multisets of linear formulas. As we will soon see, $\text{Pr}(F)$ is closed under the double negation operation for any F .

These are the basic design principles of phase semantics: interpreting linear formulas as closed subsets of a monoid for the closure operation induced by the orthogonality relation *w.r.t.* a specific subset \perp of the monoid which is an *abstraction of the provable sequents*.

► **Definition 2.** A **phase space** is a 4-tuple $\mathcal{M} = (M, 1, \cdot, \perp)$ where $(M, 1, \cdot)$ is a commutative monoid and $\perp \subseteq M$. For $X, Y \subseteq M$, define the following operations: $XY := \{x \cdot y \mid x \in X, y \in Y\}$ and $X^\perp := \{y \mid \forall x \in X, x \cdot y \in \perp\}$.

Facts are those $X \subseteq M$ such that $X = X^{\perp\perp}$. (Equivalently, $X = Y^\perp$ for some $Y \subseteq M$.)

► **Example 3.** Consider the additive monoid $(\mathbb{Z}, 0, +)$ and let $\perp = \{0\}$. For any set $S \subseteq \mathbb{Z}$, $S^\perp = \{y \mid \forall x \in S, x + y = 0\}$. Therefore, if S is not singleton then $S^\perp = \emptyset$; so, $S^{\perp\perp} = \mathbb{Z}$. On the other hand, $\{x\}^\perp = \{-x\}$. The facts of this phase space are \emptyset , singleton sets, and \mathbb{Z} .

► **Proposition 4.** Let $X, Y \subseteq M$. Then the following properties hold.

- | | |
|---|--|
| 1. $X \subseteq Y^\perp \iff XY \subseteq \perp$ | 4. $X \subseteq X^{\perp\perp}$ |
| 2. $XX^\perp \subseteq \perp$ | 5. $X^{\perp\perp\perp} = X^\perp$ |
| 3. $X \subseteq Y \implies Y^\perp \subseteq X^\perp$ | 6. $(X \cup Y)^\perp = X^\perp \cap Y^\perp$ |

Let X and Y be facts. We define the following operations on facts.

$$\begin{aligned} X \otimes Y &:= (XY)^{\perp\perp} & X \wp Y &:= (X^\perp Y^\perp)^\perp \\ X \& Y &:= X \cap Y & X \oplus Y &:= (X \cup Y)^{\perp\perp} \end{aligned}$$

► **Proposition 5.** Let X, Y be facts. Then, $1 \in X \wp Y \iff X^\perp \subseteq Y$.

Fix a phase space \mathcal{M} and let \mathcal{X} be its set of facts. Fix $V : \mathcal{A} \rightarrow \mathcal{X}$ where \mathcal{A} is the set of atoms. A phase space along with such a valuation V is called a **phase model**. The semantics $\llbracket F \rrbracket$ of a MALL formula F is parameterised by a valuation (suppose, V) which we will denote by $\llbracket F \rrbracket^V$. We are now ready to define the semantics which is defined inductively as follows:

$$\begin{aligned} \llbracket a \rrbracket^V &= V(a) & \llbracket a^\perp \rrbracket^V &= \llbracket a \rrbracket^{V^\perp} & a &\in \mathcal{A} \\ \llbracket \mathbf{1} \rrbracket^V &= \{1\}^{\perp\perp} & \llbracket \perp \rrbracket^V &= \perp \\ \llbracket \mathbf{0} \rrbracket^V &= \{\emptyset\}^{\perp\perp} & \llbracket \top \rrbracket^V &= M \\ \llbracket F_1 \odot F_2 \rrbracket^V &= \llbracket F_1 \rrbracket^V \odot \llbracket F_2 \rrbracket^V & \odot &\in \{\otimes, \wp, \&, \oplus\} \end{aligned}$$

When V is clear from the context, we shall drop it, simply writing $\llbracket F \rrbracket$. Finally, we generalise the interpretation to sequents of the form $\Gamma = F_1, \dots, F_n$ as $\llbracket \Gamma \rrbracket = \llbracket F_1 \wp F_2 \wp \dots \wp F_n \rrbracket$.

► **Theorem 6** (MALL Soundness [30]). *If MALL $\vdash \Gamma$ then for all phase models (\mathcal{M}, V) , $1 \in \llbracket \Gamma \rrbracket^V$.*

► **Example 7.** To illustrate the utility of the phase semantics, we show that in any provable multiplicative formula F (i.e. a MALL formula with only multiplicative connectives), an atom occurs exactly as many times as its negation. Fix an arbitrary atom a occurring in F . Let $\llbracket F \rrbracket^V$ be the interpretation of F in the phase space in Example 3 w.r.t. the valuation V that maps the atom a to $\{1\}$ and every other atom to $\{0\}$. In this phase space, it is easy to see that $X \otimes Y = X \wp Y = \{x + y \mid x \in X, y \in Y\}$. By Theorem 6, if F is provable, $0 \in \llbracket F \rrbracket^V$ hence number of occurrences of a in F is equal to the number of occurrences of a^\perp .

Note that a syntactic proof would require the heavy tool of MALL cut-admissibility.

- **Definition 8.** *The syntactic model $(\text{MALL}^\bullet, \emptyset, \cdot, \perp, V)$ is a phase model such that:*
- $(\text{MALL}^\bullet, \emptyset, \cdot)$, called **syntactic monoid**, is the free commutative monoid generated by all formulas. In other words, MALL^\bullet is the set of all sequents construed as finite multisets, the empty multiset \emptyset is the monoid identity, and multiset union is the monoid operation.
 - $\perp = \text{Pr}(\perp)$ i.e. \perp is set of all provable sequents.
 - $V(a) = \text{Pr}(a)$ for all atoms $a \in \mathcal{A}$.

► **Remark 9.** Note that for the syntactic model to be well-defined one needs to show that $\text{Pr}(a)$ is a fact in the phase space $(\text{MALL}^\bullet, \emptyset, \cdot, \perp, V)$.

► **Lemma 10** (Adequation Lemma for MALL). *For all formulas F , $\llbracket F \rrbracket^V \subseteq \text{Pr}(F)$.*

► **Theorem 11** (MALL Completeness [30]). *If for all phase models (\mathcal{M}, V) , $1 \in \llbracket \Gamma \rrbracket^V$ then MALL $\vdash \Gamma$.*

Proof sketch. Suppose for any phase model (\mathcal{M}, V) , $1 \in \llbracket \Gamma \rrbracket^V$. In particular, this holds for the syntactic model. By Lemma 10, $\llbracket \Gamma \rrbracket^V \subseteq \text{Pr}(\Gamma)$ (construing Γ as a parr formula). Therefore, $\emptyset \in \text{Pr}(\Gamma)$ (recall \emptyset is the unit of syntactic monoid). Hence, $\vdash \Gamma$. ◀

Okada [42] observed that one can obtain the cut-admissibility of MALL for free by slightly modifying the definition of the syntactic monoid. Now define $\perp = \text{Pr}_{cf}(\perp)$ and $V(a) = \text{Pr}_{cf}(a)$. The refined adequation lemma $\llbracket F \rrbracket \subseteq \text{Pr}_{cf}(F)$ follows exactly as in Lemma 10.

► **Theorem 12** (MALL cut-free completeness [42]). *If for any phase model (\mathcal{M}, V) , $1 \in \llbracket \Gamma \rrbracket^V$ then MALL $\vdash_{cf} \Gamma$.*

► **Corollary 13.** *MALL admits cuts.*

Proof. Suppose MALL $\vdash \Gamma$. By Theorem 6, $\emptyset \in \llbracket \Gamma \rrbracket^V$ for the syntactic model of MALL. By Theorem 12, MALL $\vdash_{cf} \Gamma$. ◀

2.2 Fixpoint theory and fixpoint logic

In this section we will recall some background on the fundamental fixpoint theorems of lattice theory. Not only will we use them several times in our technical proofs, but also, they will provide the intuition about the design of proof systems with fixpoint rules and their corresponding semantics. For the rest of this subsection, let (S, \leq_S) be a complete lattice with least element \perp and greatest element \top .

► **Theorem 14** ([33, 49]). *Let $f : S \rightarrow S$ be a monotonic function. The set of fixpoints of f is non-empty and equipped with \leq_S forms a complete lattice.*

► **Definition 15.** *Let $f : S \rightarrow S$ be a monotonic function. f is said to be **Scott-continuous** if for each directed subset S we have $f(\bigvee_{S_i \in S} S_i) = \bigvee_{S_i \in S} f(S_i)$.*

► **Theorem 16** (Kleene Fixed Point Theorem). *Every Scott-continuous function f has the least fixpoint $\bigvee_{n \in \omega} f^n(\perp)$.*

Observe that this is a constructive formulation of a fixpoint. Cousot and Cousot proved a constructive version of Theorem 14 without using the Scott-continuity hypothesis [17]. Let $f : S \rightarrow S$ be a monotonic function. The **lower iteration sequence** for f starting with $x \in S$ is the sequence $\langle \mathcal{U}_\alpha \mid \alpha \in \text{Ord} \rangle$ of elements of S defined by transfinite induction as follows: (i) $\mathcal{U}_0 = x$; (ii) $\mathcal{U}_{\alpha+1} = f(\mathcal{U}_\alpha)$; and, (iii) $\mathcal{U}_\lambda = \bigwedge_{\alpha < \lambda} \mathcal{U}_\alpha$ for λ a limit ordinal.

► **Theorem 17** ([17]). *Let $f : S \rightarrow S$ be a monotonic function. The lower iteration sequence for f starting from \perp is increasing and there exists an ordinal θ (called the **closure ordinal** of f) such that $\mathcal{U}_\theta = \mathcal{U}_{\theta+1}$. Moreover, \mathcal{U}_θ is the least fixpoint of f .*

Note that one defines upper iteration sequence by taking supremums at limit ordinals. Dually, the greatest fixpoint is the stationary point of the decreasing upper iteration sequence starting from \top .

► **Remark 18.** The closure ordinal of a Scott-continuous function is at most ω .

2.3 Multiplicative additive linear logic with fixpoints

In this subsection we recall the propositional version of logic μ MALL and its wellfounded proof system μ MALL^{ind} introduced in [7]. (See Appendix A for details)

► **Definition 19.** *Fix a countable set of atoms $\mathcal{A} = \{a, b, \dots\}$ and variables $\mathcal{V} = \{x, y, \dots\}$ such that $\mathcal{A} \cap \mathcal{V} = \emptyset$. μ MALL **pre-formulas** are given by the following grammar:*

$$F, G ::= \mathbf{0} \mid \top \mid \perp \mid \mathbf{1} \mid a \mid a^\perp \mid x \mid F \wp G \mid F \otimes G \mid F \oplus G \mid F \& G \mid \mu x.F \mid \nu x.F$$

where $a \in \mathcal{A}$, $x \in \mathcal{V}$, and μ, ν bind the variable x in F . When a pre-formula is closed (i.e. no free variables), we simply call it a **formula**.

Negation, $(\bullet)^\perp$, defined as a meta-operation on pre-formulas, will be used only on formulas. As it is not part of the syntax, we do not need any positivity condition on the fixed-point expressions. As expected, least and greatest fixed points are the dual of each other.

► **Definition 20.** **Negation** of a pre-formula is defined inductively as follows.

$$\begin{aligned} \mathbf{0}^\perp &= \top; & \top^\perp &= \mathbf{0}; & \perp^\perp &= \mathbf{1}; & \mathbf{1}^\perp &= \perp; & (a)^\perp &= a^\perp; & a^{\perp\perp} &= a; \\ x^\perp &= x; & (F \wp G)^\perp &= F^\perp \otimes G^\perp; & (F \otimes G)^\perp &= F^\perp \wp G^\perp; & (F \oplus G)^\perp &= F^\perp \& G^\perp; \\ (F \& G)^\perp &= F^\perp \oplus G^\perp; & (\mu x.F)^\perp &= \nu x.F^\perp; & (\nu x.F)^\perp &= \mu x.F^\perp. \end{aligned}$$

35:8 Phase Semantics for μ MALL

In order to extend the phase semantics of MALL to μ MALL^{ind} we extend valuations to variables *i.e.* for any valuation V , $\text{dom}(V) = \mathcal{A} \cup \mathcal{V}$ and define $\llbracket F \rrbracket^V$ by induction on F with the usual interpretation of section 2.1 for atoms, units, and multiplicative-additive connectives and as follows for fixpoints formulas:

$$\llbracket \mu x.F \rrbracket^V = \bigcap_{X \in \mathcal{X}} \left\{ X \mid \llbracket F \rrbracket^{V[x \mapsto X]} \subseteq X \right\} \quad ; \quad \llbracket \nu x.F \rrbracket^V = \left(\bigcup_{X \in \mathcal{X}} \left\{ X \mid X \subseteq \llbracket F \rrbracket^{V[x \mapsto X]} \right\} \right)^{\perp\perp}$$

$$\text{where } V[x \mapsto X](y) := \begin{cases} V(y) & \text{if } y \neq x; \\ X & \text{if } y = x. \end{cases}$$

However completeness fails for such an interpretation. Indeed, not all facts necessarily have a pre-image, therefore $\llbracket F \rrbracket^{V[x \mapsto X]}$ does not exactly correspond to syntactic substitution and *the tentative syntactic model is not a phase model*. We need to allow strict subsets of \mathcal{X} for building fixpoints. Obviously, one cannot consider any subsets of \mathcal{X} for this purpose and we shall require that they satisfy some closure properties. Therefore we restrict the codomain of $\llbracket \bullet \rrbracket^V$ to subspaces of \mathcal{X} closed under μ MALL operations. For any set of facts $\mathcal{D} \subseteq \mathcal{X}$, the set of contexts is given by the following grammar where $\odot \in \{\otimes, \wp, \&, \oplus\}$. Let $\mathbb{F}_{\mathcal{D}}$ denote the set of contexts with exactly one hole.

$$f, g ::= [] \mid X \in \mathcal{D} \mid f \odot g$$

For $f \in \mathbb{F}_{\mathcal{D}}$, define $\mu f = \bigcap_{X \in \mathcal{D}} \{X \mid f(X) \subseteq X\}$ and $\nu f = \left(\bigcup_{X \in \mathcal{D}} \{X \mid X \subseteq f(X)\} \right)^{\perp\perp}$.

► **Definition 26.** $\mathcal{D} \subseteq \mathcal{X}$ is said to be μ -closed if

- $\{\perp, \perp\perp, M, M^\perp\} \subseteq \mathcal{D}$;
- \mathcal{D} is closed under the operations $\otimes, \wp, \&$, and \oplus ; and
- for all $f \in \mathbb{F}_{\mathcal{D}}$, $\mu f \in \mathcal{D}$ and $\nu f \in \mathcal{D}$.

A phase space \mathcal{M} equipped with a μ -closed set of facts \mathcal{D} is called a μ -**phase space**.

A \mathcal{D} -**valuation** is a map of the form $V : \mathcal{A} \cup \mathcal{V} \rightarrow \mathcal{D}$. A μ -phase space along with a \mathcal{D} -valuation is called a μ -**phase model**. The μ -**phase semantics** $\llbracket \bullet \rrbracket$ is a function that takes a μ MALL pre-formula F and returns a fact in \mathcal{D} .

Note that $\llbracket \mu x.F \rrbracket^V$ and $\llbracket \nu x.F \rrbracket^V$ are defined as before except X ranges over \mathcal{D} . A priori, the semantics of pre-formula is only an element of \mathcal{X} . The closure properties of \mathcal{D} ensures $\llbracket F \rrbracket^V \in \mathcal{D}$ for every formula F and \mathcal{D} -valuation V .

► **Lemma 27** (Monotonicity). *Let F be a μ MALL pre-formula. If $X \subseteq Y$ then $\llbracket F \rrbracket^{V[x \mapsto X]} \subseteq \llbracket F \rrbracket^{V[x \mapsto Y]}$.*

Proof. A proof can be found in Appendix B.1.1. ◀

An application of monotonicity is showing that the interpretation of the fixpoint operators are indeed fixpoints in the mathematical sense:

► **Theorem 28.** *Let \mathcal{D} be μ -closed and $f \in \mathbb{F}_{\mathcal{D}}$. Then μf and νf are the least and greatest fixpoints of f in \mathcal{D} .*

Proof. A proof can be found in Appendix B.1.2. ◀

Note that Theorem 28 cannot be proved directly by Theorem 14 since \mathcal{D} is not necessarily a complete lattice. Moreover, it does not also imply that \mathcal{D} is a complete lattice by the converse of Theorem 14 since we show that it has fixpoints of a particular kind of monotonic function, not any arbitrary monotonic function. Given a valuation V , define

$$V^\perp(p) = \begin{cases} V(p) & \text{if } p \in \mathcal{A}; \\ V(p)^\perp & \text{if } p \in \mathcal{V}. \end{cases}$$

We have $V^{\perp\perp} = V$ and $V[x \mapsto X]^\perp = V^\perp[x \mapsto X^\perp]$.

► **Lemma 29** (Duality preservation). *For any μ MALL preformula F , $\llbracket F^\perp \rrbracket^{V^\perp} = (\llbracket F \rrbracket^V)^\perp$.*

Proof. A proof can be found in Appendix B.1.3. ◀

3.1 Soundness

► **Theorem 30** (μ MALL^{ind} Soundness). *If μ MALL^{ind} $\vdash \Gamma$ then for all μ -phase models $(\mathcal{M}, \mathcal{D}, V)$, $1 \in \llbracket \Gamma \rrbracket^V$.*

Proof. Fix an arbitrary μ -phase model $(\mathcal{M}, \mathcal{D}, V)$. Given a proof π of $\vdash \Gamma$ we will induct on π . The proof is similar to that of Theorem 6 except for the fixpoint cases. Suppose the last rule of π is a (μ) rule. We have that $\Gamma = \Gamma', \mu x.F$. Assume that we have proved $\llbracket F(\mu x.F) \rrbracket^V \subseteq \llbracket \mu x.F \rrbracket^V$. We have the following:

$$\begin{aligned} & (\llbracket \mu x.F \rrbracket^V)^\perp \subseteq (\llbracket F(\mu x.F) \rrbracket^V)^\perp && \text{[Proposition 4.3]} \\ \Rightarrow & (\llbracket \Gamma' \rrbracket^V)^\perp \cdot (\llbracket \mu x.F \rrbracket^V)^\perp \subseteq (\llbracket \Gamma' \rrbracket^V)^\perp \cdot (\llbracket F(\mu x.F) \rrbracket^V)^\perp \\ \Rightarrow & \left((\llbracket \Gamma' \rrbracket^{V^\perp} \cdot \llbracket F(\mu x.F) \rrbracket^{V^\perp})^\perp \right) \subseteq \left((\llbracket \Gamma' \rrbracket^{V^\perp} \cdot \llbracket \mu x.F \rrbracket^{V^\perp})^\perp \right) && \text{[Proposition 4.3]} \\ \Leftrightarrow & \llbracket \Gamma' \wp F(\mu x.F) \rrbracket^V \subseteq \llbracket \Gamma' \wp \mu x.F \rrbracket^V \\ \Rightarrow & 1 \in \llbracket \Gamma' \wp \mu x.F \rrbracket^V && \text{[IH]} \end{aligned}$$

Therefore, it suffices to prove $\llbracket F(\mu x.F) \rrbracket^V \subseteq \llbracket \mu x.F \rrbracket^V$. Observe that $\llbracket F(\mu x.F) \rrbracket^V = \llbracket F \rrbracket^{V[x \mapsto \llbracket \mu x.F \rrbracket^V]}$. Let $X \in \mathcal{D}$ such that $\llbracket F \rrbracket^{V[x \mapsto X]} \subseteq X$ (we thus have $\llbracket \mu x.F \rrbracket \subseteq X$). We need to show that $\llbracket F \rrbracket^{V[x \mapsto \llbracket \mu x.F \rrbracket^V]} \subseteq X$. It suffices to show that $\llbracket F \rrbracket^{V[x \mapsto \llbracket \mu x.F \rrbracket^V]} \subseteq \llbracket F \rrbracket^{V[x \mapsto X]}$ which is true by Lemma 27.

Now suppose the last rule is a (ν) rule i.e. $\Gamma = \Gamma', \nu x.F$ such that the coinductive invariant is S . We need to show that $1 \in \llbracket \Gamma' \wp \nu x.F \rrbracket^V$ which by Proposition 5 is equivalent to showing $\llbracket \Gamma' \rrbracket^{V^\perp} \subseteq \llbracket \nu x.F \rrbracket^V$. By hypothesis, we have that $1 \in \llbracket \Gamma' \wp S \rrbracket^V$ which is similarly equivalent to $\llbracket \Gamma' \rrbracket^{V^\perp} \subseteq \llbracket S \rrbracket^V$. Therefore it suffices to show that $\llbracket S \rrbracket^V \subseteq \llbracket \nu x.F \rrbracket^V$:

$$\begin{aligned} & 1 \in \llbracket S^\perp \wp F(S) \rrbracket^V && \text{[IH]} \\ \Leftrightarrow & (\llbracket S^\perp \rrbracket^V)^\perp \subseteq \llbracket F(S) \rrbracket^V && \text{[Proposition 5]} \\ \Leftrightarrow & \llbracket S \rrbracket^V \subseteq \llbracket F(S) \rrbracket^V = \llbracket F \rrbracket^{V[x \mapsto \llbracket S \rrbracket^V]} && \text{[Lemma 29]} \\ \Rightarrow & \llbracket S \rrbracket^V \subseteq \llbracket \nu x.F \rrbracket^V && \text{◀} \end{aligned}$$

3.2 Completeness

Completeness for fixpoint logics are generally quite difficult since analyticity does not guarantee a subformula property. One is faced with a similar *cul de sac* in proving the cut-elimination of μ MALL^{ind} since it is not straightforward to define the notion of the complexity

of the cut formula which reduces with each step of cut-elimination. This problem is solved in [5] by invoking a technique similar to the Tait-Girard reducibility candidates (originally formulated to establish certain properties of various typed lambda calculi [48, 29]). Recall that the completeness of phase semantics gives cut admissibility for free. Therefore, it is not surprising that in order to prove completeness one needs to invoke reducibility candidates.

► **Definition 31.** Let $(\mathcal{M}, \mathcal{D}, V)$ be a μ -phase model. Given a μ MALL formula F , the **reducibility candidates** of F , denoted $\langle F \rangle$, is given by $\{X \in \mathcal{X} \mid F^\perp \in X \subseteq \text{Pr}_{cf}(F)\}$.

► **Proposition 32.** $X \in \langle F \rangle \iff X^\perp \in \langle F^\perp \rangle$

Proof. We first note that $\text{Pr}_{cf}(\bullet)$ can be straightforwardly generalised to sets of formulas as follows: $\text{Pr}_{cf}(\{F_1, \dots, F_n\}) = \bigcup_{i \in [n]} \text{Pr}_{cf}(F_i)$. Let $X \in \langle F \rangle$. Then $\{F^\perp\} \subseteq X \implies X^\perp \subseteq \{F^\perp\}^\perp = \text{Pr}_{cf}(F^\perp)$. Also, $X \subseteq \text{Pr}_{cf}(F) \implies \text{Pr}_{cf}(F)^\perp = \text{Pr}_{cf}(\text{Pr}_{cf}(F)) \subseteq X$. But $F \in \text{Pr}_{cf}(\text{Pr}_{cf}(F))$. Hence done. ◀

We are now ready to define the μ -syntactic model. Recall that $\text{Pr}_{cf}(F)$ is the set of all sequents Γ such that $\vdash \Gamma, F$ is *cut-free provable*.

► **Definition 33.** The μ -syntactic model, denoted $(\mu\text{MALL}^\bullet, \emptyset, \cdot, \perp, V)$, is defined as:

- $(\mu\text{MALL}^\bullet, \emptyset, \cdot)$ is the free commutative monoid generated by all formulas.
- $\perp = \text{Pr}_{cf}(\perp)$.
- $V(p) = \text{Pr}_{cf}(p)$ for all $p \in \mathcal{A} \cup \mathcal{V}$.
- $\mathcal{D} = \bigcup_{F \in \text{Form}} \langle F \rangle$ where *Form* is the set of all μ MALL formulas.

Observe that $\perp = \text{Pr}_{cf}(\perp) \in \mathcal{D}$ and that \mathcal{D} indeed contains \perp^\perp , μMALL^\bullet and $\mu\text{MALL}^{\bullet\perp}$.

► **Lemma 34** (Adequation Lemma for $\mu\text{MALL}^{\text{ind}}$). Let $F(\bar{x})$ be a pre-formula, $\bar{G} \equiv G_1, \dots, G_m$ with $\bar{x} \equiv x_1, \dots, x_m$ be an m -tuple of pre-formulas and let $\bar{X} \equiv X_1 \dots X_m$ be an m -tuple of facts such that $X_i \in \langle G_i \rangle$. We have $\llbracket F \rrbracket^{V[\bar{x} \mapsto \bar{X}]} \subseteq \text{Pr}_{cf}(F(\bar{G}/\bar{x}))$.

Proof. By induction on F . Multiplicative additive cases are treated as in the proof of Lemma 10; we detail fixed-point cases only.

Case 1. Suppose $F = \mu y.F'$. Let $\xi = \mu y.F'(\bar{G}/\bar{x})$. In the proof of Theorem 30, we showed that for any formula F_0 , $\text{Pr}_{cf}(F_0(\mu x.F_0/x)) \subseteq \text{Pr}_{cf}(\mu x.F_0)$. Therefore, $\text{Pr}_{cf}(F'(\bar{G}/\bar{x}, \xi/y)) \subseteq \text{Pr}_{cf}(\xi)$. Let $Y^* = \llbracket F' \rrbracket^{V[\bar{x} \mapsto \bar{X}, y \mapsto \text{Pr}_{cf}(\xi)]}$. By induction hypothesis and that fact that $\text{Pr}_{cf}(F_0) \in \langle F_0 \rangle$ for all formulas F_0 , we have the following.

$$Y^* \subseteq \text{Pr}_{cf}(F'(\bar{G}/\bar{x}, \xi/y)) \tag{1}$$

Therefore, it is enough to show that $\llbracket \mu y.F' \rrbracket^{V[\bar{x} \mapsto \bar{X}]} \subseteq Y^*$. Take $\Gamma \in \llbracket \mu y.F' \rrbracket^{V[\bar{x} \mapsto \bar{X}]}$. For any fact $Y \in \mathcal{D}$, to show $\Gamma \in Y$ it is enough to show $\llbracket F' \rrbracket^{V[\bar{x} \mapsto \bar{X}, y \mapsto Y]} \subseteq Y$. Therefore we need to check that $\llbracket F' \rrbracket^{V[\bar{x} \mapsto \bar{X}, y \mapsto Y^*]} \subseteq Y^* = \llbracket F' \rrbracket^{V[\bar{x} \mapsto \bar{X}, y \mapsto \text{Pr}_{cf}(\xi)]}$. This follows by Lemma 27 from 1.

Case 2. Suppose $F = \nu y.F'$. For any fact Y , define $Z_Y = \llbracket F' \rrbracket^{V[\bar{x} \mapsto \bar{X}, y \mapsto Y]}$. Let $\Gamma \in \bigcup \{Y \in \mathcal{D} \mid Y \subseteq Z_Y\}$. Therefore, there exists, $Y^* \in \mathcal{D}$ such that $\Gamma \in Y^* \subseteq Z_{Y^*}$. Since $Y^* \in \mathcal{D}$, $Y^* \in \langle \xi \rangle$ for some formula ξ . By induction hypothesis, $\llbracket F' \rrbracket^{V[\bar{x} \mapsto \bar{X}, y \mapsto Y^*]} \subseteq \text{Pr}_{cf}(F'(\bar{G}/\bar{x}, \xi/y))$.

We will now show that $\text{Pr}_{cf}(F'(\bar{G}/\bar{x}, \xi/y)) \subseteq \text{Pr}_{cf}(F(\bar{G}/\bar{x}))$. Let $\Delta \in \text{Pr}_{cf}(F'(\bar{G}/\bar{x}, \xi/y))$. If we show that $\xi^\perp \in \text{Pr}_{cf}(F'(\bar{G}/\bar{x}, \xi/y))$, we have the following.

$$\frac{\frac{\vdash \xi^\perp, F'(\overline{G}/\overline{x}, \xi/y)}{\vdash \Delta, F'(\overline{G}/\overline{x}, \xi/y)} \quad \frac{\vdash (F'(\overline{G}/\overline{x}, \xi/y))^\perp, F'(\overline{G}/\overline{x}, F'(\overline{G}/\overline{x}, \xi/y)/y)}{\vdash \Delta, \nu x. F'(\overline{G}/\overline{x})} \text{ (func)}}{\vdash \Delta, \nu x. F'(\overline{G}/\overline{x})} \text{ (\nu)}$$

In order to show $\xi^\perp \in \text{Pr}_{cf}(F'(\overline{G}/\overline{x}, \xi/y))$, we use the induction hypothesis to reduce the problem to showing $\xi^\perp \in \llbracket F' \rrbracket^{V[\overline{x} \mapsto \overline{X}, y \mapsto Y^*]}$ which is true since $Y^* \in \langle \xi \rangle$. Therefore we have,

$$\begin{aligned} \Gamma \in \text{Pr}_{cf}(\nu y. F'(\overline{G}/\overline{x})) &\Rightarrow \bigcup \{Y \in \mathcal{D} \mid Y \subseteq Z_Y\} \subseteq \text{Pr}_{cf}(\nu y. F'(\overline{G}/\overline{x})) \\ &\Rightarrow \left(\bigcup \{Y \in \mathcal{D} \mid Y \subseteq Z_Y\} \right)^{\perp\perp} \subseteq \text{Pr}_{cf}(\nu y. F'(\overline{G}/\overline{x})) \end{aligned}$$

This concludes our proof. \blacktriangleleft

► **Lemma 35.** *Using notations of the previous lemma, we have that $F^\perp(\overline{G}/\overline{x}) \in \llbracket F \rrbracket^{V[\overline{x} \mapsto \overline{X}]}$.*

Proof. Observe that Lemma 34 and Proposition 32 imply $\llbracket F^\perp \rrbracket^{V[\overline{x} \mapsto \overline{X}^\perp]} \subseteq \text{Pr}_{cf}(F^\perp(\overline{G}^\perp/\overline{x}))$. Therefore, $\{F^\perp(\overline{G}^\perp/\overline{x})\} \llbracket F^\perp \rrbracket^{V[\overline{x} \mapsto \overline{X}^\perp]} \subseteq \{F^\perp(\overline{G}^\perp/\overline{x})\} \cdot \text{Pr}_{cf}(F^\perp(\overline{G}^\perp/\overline{x})) \subseteq \text{Pr}_{cf}(\perp) = \perp$. By Proposition 4.1, $F^\perp(\overline{G}^\perp/\overline{x}) \subseteq \left(\llbracket F^\perp \rrbracket^{V[\overline{x} \mapsto \overline{X}^\perp]} \right)^\perp$ which is $\llbracket F \rrbracket^{V[\overline{x} \mapsto \overline{X}]}$ by Lemma 29. \blacktriangleleft

Observe that by Lemma 34 and Lemma 35 we have $\llbracket F \rrbracket^{V[x \mapsto X]} \in \langle F(\overline{G}/\overline{x}) \rangle$. This ensures that \mathcal{D} is μ -closed. Consequently, $(\mu\text{MALL}^\bullet, \emptyset, \cdot, \perp, V)$ is a μ -phase space model.

► **Theorem 36** ($\mu\text{MALL}^{\text{ind}}$ cut-free completeness). *If for any μ -phase model $(\mathcal{M}, \mathcal{D}, V)$, $1 \in \llbracket \Gamma \rrbracket^V$ then $\mu\text{MALL}^{\text{ind}} \vdash_{cf} \Gamma$.*

► **Corollary 37.** $\mu\text{MALL}^{\text{ind}}$ admits cuts.

► **Remark 38.** Note that this is an alternate proof of Theorem 25.

3.3 Closure ordinals

Closure ordinals are a standard measure of the complexity of any (class of) monotone functions. The closure ordinal of a fixed-point formula is essentially the closure ordinal of the corresponding monotone function in the truth semantics. In [21], the closure ordinal is construed as a function of the size of the finite model. The study of closure ordinals of modal logic formulas is a young and exciting area of research [18, 2, 31]. It departs from the previous notion of closure ordinals in its model-independence. In this case, closure ordinal really serves as a measure of complexity of a formula.

► **Definition 39.** *Let F be a pre-formula such that $x \in \text{fv}(F)$ and let $\mathbb{M} = (\mathcal{M}, \mathcal{D}, V)$ be a μ -phase model. We define the **closure ordinal** of F with respect to x and \mathbb{M} , denoted $\mathcal{O}_{\mathbb{M}}(F)$, as the closure ordinal of $\lambda X. \llbracket F \rrbracket^{V[x \mapsto X]}$. The closure ordinal of F with respect to x (across all models) is defined as $\mathcal{O}(F) := \sup_{\mathbb{M}} \{\mathcal{O}_{\mathbb{M}}(F)\}$. Finally, F is said to be **constructive** if its closure ordinal is at most ω .*

For any pre-formula F and phase model \mathbb{M} , $\mathcal{O}_{\mathbb{M}}(F)$ exists since $\lambda X. \llbracket F \rrbracket^{V[x \mapsto X]}$ is monotonic. Consequently, the supremum $\mathcal{O}(F)$ exists since the class of all μ -phase models is indeed a set.

► **Example 40.** Consider the pre-formula $a\wp x$. Let $\langle \mathcal{U}_\alpha^{a\wp x} \mid \alpha \in \text{Ord} \rangle$ be the iteration sequence of its approximations. $\mathcal{U}_0^{a\wp x} = \emptyset^{\perp\perp}$ and $\mathcal{U}_1^{a\wp x} = \llbracket a\wp x \rrbracket^{V[x \mapsto \mathcal{U}_0^{a\wp x}]} = V(a) \cap \emptyset^{\perp\perp}$. Therefore, $\mathcal{O}(a\wp x) = 0$.

In the tradition of μ -calculus, the name “constructive” is used loosely here, motivated by the observation that if $\mathcal{O}(F)$ is a finite ordinal strictly below ω for any pre-formula F , then $\mu x.F$ is provably equivalent to $F^{\mathcal{O}(F)}(0)$. Therefore, the class of μ MALL formulas with closure ordinal strictly less than ω can be embedded in MALL and enjoys several good properties like finite model property and decidability. Observe that if the interpretation of a formula in any phase model is Scott-continuous, then it is constructive by Theorem 16. The converse does not hold in general. In the following section we consider a proof system of μ MALL where fixed points are approximated by their ω th approximation.

4 μ MALL $_\omega$: an infinitary proof system

For a constructive formula F ,

$$\llbracket \mu x.F \rrbracket^V = \left(\bigcup_{n \geq 0} \llbracket F^n(\mathbf{0}) \rrbracket^V \right)^{\perp\perp} \quad ; \quad \llbracket \nu x.F \rrbracket^V = \bigcap_{n \geq 0} \llbracket F^n(\top) \rrbracket^V \quad (2)$$

Therefore, syntactically, $\mu x.F$ (respectively $\nu x.F$) is equivalent to an infinitary \oplus -formula $\bigoplus_{n \in \omega} F^n(\mathbf{0})$ (respectively, an infinitary $\&$ -formula $\&_{n \in \omega} F^n(\top)$). We enrich the language of μ MALL with the operators μ^n and ν^n for all $n \in \omega$. We call this language as well as the corresponding proof system, μ MALL $_\omega$. (See Appendix A for details.)

► **Definition 41.** A μ MALL $_\omega$ proof is a wellfounded (possibly infinitely branching) tree generated from the inference rules of MALL given in Figure 1 and the following rules for fixpoint operators where $\eta \in \{\mu, \nu\}$.

$$\frac{\vdash \Gamma, \mathbf{0}}{\vdash \Gamma, \mu^0 x.F} (\mu^0) \quad ; \quad \frac{\vdash \Gamma, F(\eta^n x.F/x)}{\vdash \Gamma, \eta^{n+1} x.F} (\eta^{n+1}) \quad ; \quad \frac{\vdash \Gamma, \mu^n x.F}{\vdash \Gamma, \mu x.F} (\mu^\omega)$$

$$\frac{\vdash \Gamma, \top}{\vdash \Gamma, \nu^0 x.F} (\nu^0) \quad ; \quad \frac{\vdash \Gamma, \nu^0 x.F \quad \vdash \Gamma, \nu^1 x.F \quad \vdash \Gamma, \nu^2 x.F \quad \dots}{\vdash \Gamma, \nu x.F} (\nu^\omega)$$

► **Example 42.** Let $H = (\mu x.a\wp x)^\perp \wp (a^{\perp p} \wp \mathbf{0})$ for some $p \in \omega$.

$$\frac{\frac{\frac{\frac{\frac{\frac{\vdash \nu^n y.F, \mu^{2n} x.a\wp x}{\vdash \nu^n y.F, \mu x.a\wp x} (\mu_{2n}^\omega)}{\vdash \nu^n y.F, \mu x.a\wp x} (\nu^\omega)}{\vdash \mu^p x.a^\perp \wp x, (a^{\perp p} \wp \mathbf{0})^\perp} (\otimes)}{\vdash \mu^p x.a^\perp \wp x, \nu y.F, H^\perp} (\otimes)}{\vdash a^\perp, a} (\text{id})}{\vdash a^\perp, \mu^p x.a^\perp \wp x, a \otimes (\nu y.F), H^\perp} (\wp)}{\vdash a^\perp \wp (\mu^p x.a^\perp \wp x), a \otimes (\nu y.F), H^\perp} (\mu^{p+1})}{\vdash \mu^{p+1} x.a^\perp \wp x, a \otimes (\nu y.F), H^\perp} (\mu_{p+1}^\omega)}{\vdash \Gamma_0, H^\perp} (\mu_{p+1}^\omega)$$

It is easy to show that for all $p, n \in \omega$, $\vdash \mu^p x.a^\perp \wp x, (a^{\perp p} \wp \mathbf{0})^\perp$ and $\vdash \nu^n y.F, \mu^{2n} x.a\wp x$ are provable by induction on p and n respectively.

Note that the set of inference rules (as schema) is infinite since $\{\mu_n^\omega\}_{n \in \omega}$ is a collection of ω -many rules. Such infinitary systems (called Tait-style systems) where proof trees are wellfounded with possible infinite branching are studied in various areas of logic *viz.* arithmetic [15, 41] and fixpoint logics [35, 32]. It is quite tricky to define a proper notion of the complexity of a fixpoint formula in such settings. Closely based on [32], our notion of the rank of a μMALL_ω formula is a finite sequences of ordinals.

First we will set up some notion. If $\alpha_1, \dots, \alpha_n$ are ordinals, we write $\langle \alpha_1, \dots, \alpha_n \rangle$ for the sequence σ whose length $|\sigma|$ is n and whose i th component σ_i is the ordinal α_i . Let $<_{lex}$ be the strict lexicographical ordering of finite sequences of ordinals and \leq_{lex} its reflexive closure. Note that $<_{lex}$ is a well-ordering on any set of sequences of bounded lengths but not a well-ordering in general. In particular, $\langle 1 \rangle, \langle 0, 1 \rangle, \langle 0, 0, 1 \rangle, \dots$ is an infinite descending chain in $<_{lex}$. Given two finite sequences of ordinals σ, τ , we define the component-wise ordering \leq as σ, τ iff $|\sigma| \leq |\tau|$ and $(\sigma)_i \leq (\tau)_i$ for all $1 \leq i \leq |\sigma|$. Clearly, the relation \leq is transitive. We denote the standard concatenation of sequences by $*$. Finally we define a component-wise maximum operation \sqcup by setting: (i) $\sigma \sqcup \langle \rangle := \langle \rangle$; (ii) if $\sigma = \langle b_1, \dots, b_m \rangle$ and $\tau = \langle b'_1, \dots, b'_n \rangle$, then

$$\sigma \sqcup \tau = \begin{cases} \langle \max(b_1, b'_1), \dots, \max(b_m, b'_m), b'_{m+1}, \dots, b'_n \rangle & \text{if } m \leq n; \\ \langle \max(b_1, b'_1), \dots, \max(b_n, b'_n), b_{n+1}, \dots, b_m \rangle & \text{otherwise.} \end{cases}$$

Now we are ready to define the rank of a μMALL_ω formula. The rank of every μMALL_ω formula will be a finite sequence of ordinals less than or equal to ω .

► **Definition 43.** The rank of a μMALL_ω formula F , denoted $\text{rk}(F)$, is defined by induction on F as follows.

- if F is an atom, a variable, or a unit, then $\text{rk}(F) = \langle 0 \rangle$;
- if $F = G \odot G'$, then $\text{rk}(F) = (\text{rk}(G) \sqcup \text{rk}(G')) * \langle 0 \rangle$ where $\odot \in \{\wp, \otimes, \oplus, \&\}$;
- if $F = \eta^n x.G$, then $\text{rk}(F) = \text{rk}(G) * \langle n \rangle$ where $\eta \in \{\mu, \nu\}$;
- if $F = \eta x.G$, then $\text{rk}(F) = \text{rk}(G) * \langle \omega \rangle$ where $\eta \in \{\mu, \nu\}$.

► **Example 44.** Consider $H = (\nu x.a^\perp \otimes x)\wp(a^p \wp \mathbf{0})$ from Example 42.

$$\begin{aligned} \text{rk}(H) &= \text{rk}(\nu x.a^\perp \otimes x) \sqcup \text{rk}(a^p \wp \mathbf{0}) * \langle 0 \rangle = \left((\text{rk}(a^\perp \otimes x) * \langle \omega \rangle) \sqcup \overbrace{\langle 0, \dots, 0 \rangle}^{p+1} \right) * \langle 0 \rangle \\ &= \left(\langle 0, 0, \omega \rangle \sqcup \overbrace{\langle 0, \dots, 0 \rangle}^{p+1} \right) * \langle 0 \rangle = \langle 0, 0, \omega, \overbrace{0, \dots, 0}^{\max(1, p-1)} \rangle \end{aligned}$$

► **Lemma 45.** Let F be a μMALL_ω pre-formula such that $x \in \text{fv}(F)$. Let ξ be a preformula such that $\text{rk}(F) \leq \text{rk}(\xi)$. Then, there exists a finite (possibly empty) sequence of ordinals σ such that $\text{rk}(F(\xi/x)) = \text{rk}(\xi) * \sigma$.

Proof. A proof can be found in Appendix B.2.1. ◀

► **Theorem 46.** The following hold for any μMALL_ω formula F :

1. $\text{rk}(F) <_{lex} \text{rk}(F \odot G)$ and $\text{rk}(G) <_{lex} \text{rk}(F \odot G)$;
2. $\text{rk}(\mathbf{0}) <_{lex} \text{rk}(\mu^0 x.F)$, $\text{rk}(\top) <_{lex} \text{rk}(\nu^0 x.F)$;
3. $\text{rk}(F(\eta^n x.F/x)) <_{lex} \text{rk}(\eta^{n+1} x.F)$ for all $n < \omega$;
4. $\text{rk}(\eta^n x.F) <_{lex} \text{rk}(\eta x.F)$ for all $n < \omega$.

Proof. The first, second, and fourth assertions are immediate from Definition 43. For the third one, we have two cases:

- Suppose $x \notin \text{fv}(F)$. Then, $F(\eta^n x.F/x) = F$ and the result follows from Definition 43.
- Otherwise, note that we have $\text{rk}(F) \trianglelefteq \text{rk}(\eta^n x.F)$. By Lemma 45, $\text{rk}(F(\eta^n x.F/x)) = \text{rk}(\eta^n x.F) * \sigma = \text{rk}(F) * \langle n \rangle * \sigma$ for some σ . But $\text{rk}(\eta^{n+1} x.F) = \text{rk}(F) * \langle n+1 \rangle$.

Hence we are done. \blacktriangleleft

► **Definition 47.** The strong closure $\text{SC}(F)$ of a μMALL_ω formula F is the least set s.t.:

- $F \in \text{SC}(F)$;
- $G \odot H \in \text{SC}(F) \implies \{G, H\} \subset \text{SC}(F)$ where $\odot \in \{\wp, \otimes, \oplus, \&\}$;
- $\mu^0 x.G \in \text{SC}(F) \implies \mathbf{0} \in \text{SC}(F)$;
- $\nu^0 x.G \in \text{SC}(F) \implies \top \in \text{SC}(F)$;
- $\eta^{n+1} x.G \in \text{SC}(F) \implies G(\eta^n x.G/x) \in \text{SC}(F)$ for all $n \in \omega$ and $\eta \in \{\mu, \nu\}$;
- $\eta x.G \in \text{SC}(F) \implies \eta^n x.G \in \text{SC}(F)$ for all $n \in \omega$ and $\eta \in \{\mu, \nu\}$.

Define F^- to be image of F under the forgetful functor that erases the explicit approximations occurring in F . For example, $(a \otimes \mu^6 x.\nu y.x \oplus y)^- = a \otimes \mu x.\nu y.x \oplus y$.

► **Theorem 48.** For any formula F , the set $\{\text{rk}(G) \mid G \in \text{SC}(F)\}$ is a well-order with respect to the $<_{lex}$ ordering.

Proof. By contradiction. Note that $|\text{rk}(G)| = |\text{rk}(G^-)|$. Furthermore, if $G \in \text{SC}(F)$ then $G^- \in \mathbb{FL}(F)$. But $\mathbb{FL}(F)$ is a finite set, therefore the set $\{|\text{rk}(G)| \mid G \in \text{SC}(F)\}$ is finite.

Assume there exists $\{\sigma_i\}_{i \in I}$ an infinite descending chain in $\{|\text{rk}(G)| \mid G \in \text{SC}(F)\}$. By the Infinite Ramsey Theorem, there is an infinite subsequence $\{\sigma_i\}_{i \in \omega}$ such that for all $j, i_j \in I$ and for all $j, j', |\sigma_{i_j}| = |\sigma_{i_{j'}}| = n$. Then, there exist $k \leq n$ and $N \in \mathbb{N}$ such that $\{(\sigma_{i_j})_k\}_{j > N}$ is a descending chain. This contradicts the wellfoundedness of natural numbers. \blacktriangleleft

4.1 Soundness and completeness of μMALL_ω

The phase semantics for μMALL_ω is much simpler to define. Like MALL the semantics can be defined given a phase space and a valuation (without the extra structure over the set of facts and extension of valuations to variables as in $\mu\text{MALL}^{\text{ind}}$). Given a phase space \mathcal{M} and a valuation function $V : \mathcal{A} \rightarrow \mathcal{X}$, we define the interpretation of multiplicative-additive connectives and units as usual, of $\mu x.F$ and $\nu x.F$ as in Equation (2), and of $\eta^n x.F$ (for $\eta \in \{\mu, \nu\}$) as follows: $\llbracket \mu^0 x.F \rrbracket = \llbracket \mathbf{0} \rrbracket$, $\llbracket \nu^0 x.F \rrbracket = \llbracket \top \rrbracket$, and $\llbracket \eta^{n+1} x.F \rrbracket = \llbracket F(\eta^n x.F/x) \rrbracket$.

► **Theorem 49** (μMALL_ω soundness). If $\vdash \Gamma$ then for all phase models (\mathcal{M}, V) , $1 \in \llbracket \Gamma \rrbracket^V$.

The proof of Theorem 49 is by a straightforward induction on the structure of the proof.

► **Lemma 50** (Adequation Lemma for μMALL_ω). For all formula F , $\llbracket F \rrbracket^V \subseteq \text{Pr}_{cf}(F)$.

Proof. By induction on $\text{rk}(F)$. The base case is when F is an atom or a unit in which case by definition $\llbracket F \rrbracket = \text{Pr}_{cf}(F)$. Suppose $F = G \odot H$ for $\odot \in \{\otimes, \wp, \&, \oplus\}$. By Theorem 46.1, $\text{rk}(G) <_{lex} \text{rk}(F)$ and $\text{rk}(H) <_{lex} \text{rk}(F)$. By IH, $\llbracket G \rrbracket^V \subseteq \text{Pr}_{cf}(G)$ and $\llbracket H \rrbracket^V \subseteq \text{Pr}_{cf}(H)$. Using standard techniques in the proof of Lemma 10 (cf. [30]), we can conclude that $\llbracket F \rrbracket = \text{Pr}_{cf}(F)$.

The case $F = \eta^0 x.G$ is trivial. Suppose $F = \eta^{n+1} x.G$ where $\eta \in \{\mu, \nu\}$. By Theorem 46.3, $\text{rk}(G(\eta^n x.G)) <_{lex} \text{rk}(\eta^{n+1} x.G)$. By IH, $\llbracket F \rrbracket = \llbracket G(\eta^n x.G) \rrbracket^V \subseteq \text{Pr}_{cf}(G(\eta^n x.G)) \subseteq \text{Pr}_{cf}(F)$.

Suppose $F = \mu x.G$. Noting that $\text{Pr}_{cf}(F)$ is a fact for all F , it is enough to show that $\bigcup_{n \geq 0} \llbracket \mu^n x.G \rrbracket \subseteq \text{Pr}_{cf}(\mu x.G)$. By Theorem 46.4, $\text{rk}(\mu^n x.G) <_{lex} \text{rk}(F)$. By IH, for all n , $\llbracket \mu^n x.G \rrbracket \subseteq \text{Pr}_{cf}(\mu^n x.G)$. Observe that $\text{Pr}_{cf}(\mu^n x.G) \subseteq \text{Pr}_{cf}(\mu x.G)$ for all n . Therefore, $\bigcup_{n \geq 0} \text{Pr}_{cf}(\mu^n x.G) \subseteq \text{Pr}_{cf}(\mu x.G)$. The case for $F = \nu x.G$ works similarly. \blacktriangleleft

► **Theorem 51** (μMALL_ω cut-free completeness). *If for any phase model (\mathcal{M}, V) , $1 \in \llbracket \Gamma \rrbracket^V$ then $\mu\text{MALL}_\omega \vdash_{cf} \Gamma$.*

► **Corollary 52.** μMALL_ω admits cuts.

Standard cut-elimination techniques for Tait-like systems employ techniques from Schütte [45] where proofs are assigned a cut rank. One shows that if there is a proof π of a sequent $\vdash \Gamma$ with cut-rank $\text{rk}(\pi) > 0$ then there is a proof π' of $\vdash \Gamma$ such that $\text{rk}(\pi') = 0$ (possibly incurring a blowup in the size of the proof). Cut-admissibility has been proved previously for Tait-style systems of fixpoints logics in [43, 14] using this technique. Semantic proofs of cut-admissibility have been explored in various logics [46, 39, 4] but, to our knowledge, this is the first semantic proof of cut-admissibility in a Tait-style system.

4.2 Finite model property

In this subsection, we show that μMALL_ω does not have the finite model property. Since MALL has finite model property, this implies that μMALL_ω cannot be embedded in MALL .

► **Lemma 53.** $\vdash \Gamma_0$ is not provable in μMALL_ω .

Proof. A proof can be found in Appendix B.2.2. ◀

► **Corollary 54.** μMALL_ω does not prove the same theorems as $\mu\text{MALL}^{\text{ind}}$.

Proof. From Example 23 obtain $\mu\text{MALL}^{\text{ind}} \vdash \Gamma_0$. The result now follows from Lemma 53. ◀

► **Theorem 55.** μMALL_ω does not have finite model property.

Proof. The proof goes by contradiction. A finite phase model (\mathcal{M}, V) has finitely many facts. Therefore, by pigeonhole principle, there exists p, q such that $p < q$ and $\llbracket a^p \wp \mathbf{0} \rrbracket^V = \llbracket a^q \wp \mathbf{0} \rrbracket^V$. Therefore, $\llbracket \mu x. a \wp x \rrbracket^V = \llbracket a^p \wp \mathbf{0} \rrbracket^V$. Consequently, $1 \in \llbracket H \rrbracket^V$ where $H = (\mu x. a \wp x)^\perp \wp (a^\perp \wp \mathbf{0})$. By Theorem 51, $\mu\text{MALL}_\omega \vdash H$. In Example 42, we show that $\mu\text{MALL}_\omega \vdash \Gamma_0, H^\perp$. By an application of the cut-rule, we have $\mu\text{MALL}_\omega \vdash \Gamma_0$ is provable. This is a contradiction by Lemma 53. ◀

5 Conclusion

In this work, we provided a sound and complete provability semantics for linear logic with fixpoints (μMALL) for the proof system with explicit induction due to Baelde and Miller [5, 7]. The completeness proof goes via an adaptation of Tait-Girard reducibility candidates. We then introduced a Tait-style proof system where fixpoints are approximated by their ω -th approximants (μMALL_ω) and we get a direct proof of completeness. Finally, we show that μMALL_ω does not have finite model property and hence is a logic with non-trivial expressivity.

We conclude by mentioning pertinent directions for future work.

- Exploring the phase semantics of μMALL° and μMALL^∞ , circular and non-wellfounded systems for μMALL respectively [6, 23], are relevant questions especially because that would help settle the Brotherston-Simpson conjecture by semantic techniques. If the conjecture is true, then it suffices to show that μMALL° is sound with respect to μ -phase models. If the conjecture is false, there are two possibilities for using phase semantics. Firstly, if we have a μMALL° theorem F which we wish to prove is not a $\mu\text{MALL}^{\text{ind}}$ theorem, then we can use the μ -phase model. Secondly, if one can devise the phase semantics of μMALL° , then the proof is reduced to checking that the space of μ -phase models and its counterpart for μMALL° are the same.

- The computation of closure ordinals in phase semantics seems quite difficult and deserves a closer study. An important problem is to compute an upper bound possibly by embedding μ MALL^{ind} in some arithmetic theory.
- Techniques involving cut ranks used to obtain cut-admissibility also provide upper bounds on the size of the cut-free proof. It would be interesting to see if Theorem 51 can be refined to obtain such bounds.
- μ MALL^{ind} has the focusing property [7] but assigning polarities to fixed point operators is not a priori clear. In fact, it holds for both possible assignments (the proofs being quite different). In the implicit case, one can syntactically argue that μ has to be positive (consequently ν should be negative). Categorical semantics of polarised μ MALL^{ind} informs us that μ should indeed be positive [25]. Can phase semantics also shed light on the polarisation of fixed points?
- We showed that μ MALL^{ind} $\not\subseteq$ μ MALL _{ω} . Noting that the Park's rule can be simulated with the (ν_ω) rule, we conjecture that μ MALL _{ω} $\not\subseteq$ μ MALL^{ind}. Moreover, μ MALL^{ind} can encode the provability of linear logic exponentials but this is not clear for μ MALL _{ω} . We conjecture that μ MALL _{ω} cannot simulate *digging*, but it can, indeed, encode *soft promotion* and *multiplexing*, thereby being at least as powerful as soft linear logic [37].

References

- 1 Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739–782. Elsevier, 1977.
- 2 Bahareh Afshari and Graham E. Leigh. On closure ordinals for the modal mu-calculus. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30–44, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2013.30.
- 3 Alfred V. Aho and Jeffrey D. Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '79, pages 110–119, New York, NY, USA, 1979. Association for Computing Machinery. doi:10.1145/567752.567763.
- 4 Jeremy Avigad. Algebraic proofs of cut elimination. *The Journal of Logic and Algebraic Programming*, 49(1):15–30, 2001. doi:10.1016/S1567-8326(01)00009-1.
- 5 David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Logic*, 13(1), January 2012. doi:10.1145/2071368.2071370.
- 6 David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.42.
- 7 David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 92–106, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 8 Stefano Berardi and Makoto Tatsuta. Classical system of martin-löf's inductive definitions is not equivalent to cyclic proof system. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 301–317, 2017.
- 9 Stefano Berardi and Makoto Tatsuta. Equivalence of inductive definitions and cyclic proofs under arithmetic. In *LICS*, pages 1–12. IEEE Computer Society, 2017.
- 10 Stefano Berardi and Makoto Tatsuta. Classical system of martin-lof's inductive definitions is not equivalent to cyclic proofs. *Log. Methods Comput. Sci.*, 15(3), 2019.
- 11 James Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions*. PhD thesis, University of Edinburgh, November 2006.

- 12 James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 51–62. IEEE, 2007.
- 13 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011.
- 14 Kai Brännler and Thomas Studer. Syntactic cut-elimination for a fragment of the modal μ -calculus. *Annals of Pure and Applied Logic*, 163(12):1838–1853, 2012. doi:10.1016/j.apal.2012.04.006.
- 15 Rudolf Carnap. *Logical Syntax of Language*. Kegan Paul, Trench and Truber, 1937.
- 16 Edmund M Clarke, Orna Grumberg, and Doron A. Peled. Model checking for the μ -calculus. In *Model checking*, chapter 7, pages 97–108. MIT Press, London, Cambridge, 1999.
- 17 Patrick Cousot and Radhia Cousot. Constructive versions of Tarski’s fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.
- 18 Marek Czarnecki. How fast can the fixpoints in modal μ -calculus be reached? In Luigi Santocanale, editor, *7th Workshop on Fixed Points in Computer Science, FICS 2010, Brno, Czech Republic, August 21-22, 2010*, pages 35–39. Laboratoire d’Informatique Fondamentale de Marseille, 2010. URL: <https://hal.archives-ouvertes.fr/hal-00512377/document#page=36>.
- 19 Ugo Dal Lago and Simone Martini. Phase semantics and decidability of elementary affine logic. *Theoretical Computer Science*, 318(3):409–433, 2004. doi:10.1016/j.tcs.2004.02.037.
- 20 Anupam Das, Abhishek De, and Alexis Saurin. Decision problems for linear logic with least and greatest fixed points. In *FSCD*, volume 228 of *LIPICs*, pages 20:1–20:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 21 Anuj Dawar and Yuri Gurevich. Fixed Point Logics. *Bulletin of Symbolic Logic*, 8(1):65–88, 2002. doi:10.2178/bsl/1182353853.
- 22 Pierre Simon de Fermat. *Oeuvres de Fermat, T.2*. Gauthier-Villars et Fils, Paris, 1894.
- 23 Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01676953>.
- 24 Thomas Ehrhard and Farzad Jafarrahmani. Categorical models of linear logic with fixed points of formulas. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470664.
- 25 Thomas Ehrhard, Farzad Jafarrahmani, and Alexis Saurin. Polarized linear logic with fixpoints (technical report). URL: <https://drive.google.com/file/d/1eQd5evwcUXYBT5f-TZbs8o4eIE4PhN9m/view?usp=sharing>.
- 26 Thomas Ehrhard, Farzad Jafarrahmani, and Alexis Saurin. On relation between totality semantic and syntactic validity. In *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*, Rome (virtual), Italy, June 2021. URL: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271408>.
- 27 Euclid. *The Elements of Euclid*. Dover Publications Inc., New York, 2nd edition, 1956. URL: https://archive.org/details/euclid_heath_2nd_ed.
- 28 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 248–262. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.248.
- 29 J. Y. Girard. Une extension de l’interprétation de Godel a l’analyse, et son application a l’élimination des coupures dans l’analyse et la théorie des types. *Proceedings of the second Scandinavian logic symposium*, 63:63–92, 1971.
- 30 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.
- 31 Maria João Gouveia and Luigi Santocanale. Aleph1 and the Modal μ -Calculus. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic*

- (*CSL 2017*), volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2017.38.
- 32 Gerhard Jäger, Mathis Kretz, and Thomas Studer. Canonical completeness of infinitary μ . *The Journal of Logic and Algebraic Programming*, 76(2):270–292, 2008. Logic and Information: From Logic to Constructive Reasoning. doi:10.1016/j.jlap.2008.02.005.
 - 33 Bronisław Knaster and Alfred Tarski. Un théorème sur les fonctions d’ensembles. *Annales de la Société Polonaise de Mathématique*, 6:133–134, 1927.
 - 34 Dexter Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982. doi:10.1016/0304-3975(82)90125-6.
 - 35 Dexter Kozen. A finite model theorem for the propositional μ -calculus. *Studia Logica*, 47(3):233–241, September 1988. doi:10.1007/BF00370554.
 - 36 Yves Lafont. The finite model property for various fragments of linear logic. *The Journal of Symbolic Logic*, 62(4):1202–1208, 1997. URL: <http://www.jstor.org/stable/2275637>.
 - 37 Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1):163–180, 2004. Implicit Computational Complexity. doi:10.1016/j.tcs.2003.10.018.
 - 38 Leonid Libkin. *Elements of Finite Model Theory*. Springer, August 2004.
 - 39 Shoji Maehara. Lattice-valued representation of the cut-elimination theorem. *Tsukuba journal of mathematics*, 15:509–521, 1991.
 - 40 Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 179–216. Elsevier, 1971. doi:10.1016/S0049-237X(08)70847-4.
 - 41 Grigori Mints. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics*, 10:548–596, 1978.
 - 42 Mitsuhiro Okada. Phase semantic cut-elimination and normalization proofs of first- and higher-order linear logic. *Theoretical Computer Science*, 227(1):333–396, 1999. doi:10.1016/S0304-3975(99)00058-4.
 - 43 Ewa Palka. An infinitary sequent system for the equational theory of *-continuous action lattices. *Fundam. Inf.*, 78(2):295–309, April 2007.
 - 44 Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2002. doi:10.1007/3-540-45931-6_25.
 - 45 Kurt Schütte. *Vollständige Systeme modaler und intuitionistischer Logik*. Springer Berlin Heidelberg, 1968. doi:10.1007/978-3-642-88664-5.
 - 46 Kurt Schütte. Syntactical and semantical properties of simple type theory. *The Journal of Symbolic Logic*, 25(4):305–326, 1960. URL: <http://www.jstor.org/stable/2963525>.
 - 47 Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300, 2017.
 - 48 William W. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium*, pages 240–251, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.
 - 49 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
 - 50 Igor Walukiewicz. *A complete deductive system for the μ -calculus*. PhD thesis, Warsaw University, 1994.
 - 51 Igor Walukiewicz. Completeness of Kozen’s axiomatisation of the propositional μ -calculus. In *LICS 95, San Diego, California, USA, June 26-29, 1995*, pages 14–24, 1995.

A Summary of μ MALL systems used in the paper

μ MALL preformulas:

$$F, G ::= \mathbf{0} \mid \top \mid \perp \mid \mathbf{1} \mid a \mid a^\perp \mid F \wp G \mid F \otimes G \mid F \oplus G \mid F \& G \\ \mid x \mid \mu x.F \mid \nu x.F$$

μ MALL^{approx}, preformulas with fixed-point approximants:

$$F, G ::= \mathbf{0} \mid \top \mid \perp \mid \mathbf{1} \mid a \mid a^\perp \mid F \wp G \mid F \otimes G \mid F \oplus G \mid F \& G \\ \mid x \mid \mu x.F \mid \nu x.F \mid \mu^n x.F \mid \nu^n x.F \quad n \in \mathbb{N}$$

	Formula language	Inference rules	Proof trees
μ MALL ^{ind}	μ MALL	Definition 22	finite
μ MALL _{ω}	μ MALL ^{approx}	Definition 41	wellfounded infinitely branching
μ MALL ^{∞}	μ MALL	Figure 1 of [6]	non-wellfounded finitely branching
μ MALL ^{\circ}	μ MALL	Figure 1 of [6]	non-wellfounded regular

► Remark. Note that, contrary to the convention used in the present paper, in [6] notation μ MALL _{ω} is used to refer to the circular proof system (that is regular non-wellfounded trees).

B Detailed proofs and clarifications

B.1 Proofs of Section 3

B.1.1 Proof of Lemma 27

Before we prove the lemma, we will prove the following claim.

▷ Claim 56. If S, T, U, V are subsets of M such that $S \subseteq T$ and $U \subseteq V$ then $SU \subseteq TV$.

Proof. Suppose $x = su \in SU$ such that $s \in S$ and $u \in U$. Then, $s \in T$ and $u \in V$. Hence $x = st \in TV$. ◁

Proof. By induction on F . The base case is when F is an atom, a variable, or a unit which are trivial.

- Suppose $F = p \in \mathcal{A} \cup \{\perp, \mathbf{1}, \mathbf{0}, \top\}$. Then, $\llbracket F \rrbracket^{V[x \mapsto X]} = \llbracket F \rrbracket^{V[x \mapsto Y]} = V(p)$.
- Suppose $F = y \in \mathcal{V}$. There are two cases. If $y \neq x$, then $\llbracket F \rrbracket^{V[x \mapsto X]} = \llbracket F \rrbracket^{V[x \mapsto Y]} = V(y)$.
Otherwise, we have $\llbracket F \rrbracket^{V[x \mapsto X]} = X \subseteq Y = \llbracket F \rrbracket^{V[x \mapsto Y]}$.

There are several subcases for the induction case.

- Suppose $F = G \otimes G'$.

$$\begin{aligned} \llbracket G \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]}; \llbracket G' \rrbracket^{V[x \mapsto X]} \subseteq \llbracket G' \rrbracket^{V[x \mapsto Y]} && \text{[By IH]} \\ \Rightarrow \llbracket G \rrbracket^{V[x \mapsto X]} \llbracket G' \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]} \llbracket G' \rrbracket^{V[x \mapsto Y]} && \text{[By Claim 56]} \\ \Rightarrow (\llbracket G \rrbracket^{V[x \mapsto X]} \llbracket G' \rrbracket^{V[x \mapsto X]})^{\perp\perp} &\subseteq (\llbracket G \rrbracket^{V[x \mapsto Y]} \llbracket G' \rrbracket^{V[x \mapsto Y]})^{\perp\perp} && \text{[By Proposition 4]} \\ \Rightarrow \llbracket F \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket F \rrbracket^{V[x \mapsto Y]} \end{aligned}$$

- Suppose $F = G \wp G'$.

$$\begin{aligned} \llbracket G \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]}; \llbracket G' \rrbracket^{V[x \mapsto X]} \subseteq \llbracket G' \rrbracket^{V[x \mapsto Y]} && \text{[By IH]} \\ \Rightarrow (\llbracket G \rrbracket^{V[x \mapsto Y]})^\perp &\subseteq (\llbracket G \rrbracket^{V[x \mapsto X]})^\perp; (\llbracket G' \rrbracket^{V[x \mapsto Y]})^\perp \subseteq (\llbracket G' \rrbracket^{V[x \mapsto X]})^\perp && \text{[By Proposition 4]} \\ \Rightarrow (\llbracket G \rrbracket^{V[x \mapsto Y]})^\perp (\llbracket G' \rrbracket^{V[x \mapsto Y]})^\perp &\subseteq (\llbracket G \rrbracket^{V[x \mapsto X]})^\perp (\llbracket G' \rrbracket^{V[x \mapsto X]})^\perp && \text{[By Claim 56]} \\ \Rightarrow \left((\llbracket G \rrbracket^{V[x \mapsto X]})^\perp (\llbracket G' \rrbracket^{V[x \mapsto X]})^\perp \right)^\perp &\subseteq \left((\llbracket G \rrbracket^{V[x \mapsto Y]})^\perp (\llbracket G' \rrbracket^{V[x \mapsto Y]})^\perp \right)^\perp && \text{[By Proposition 4]} \\ \Rightarrow \llbracket F \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket F \rrbracket^{V[x \mapsto Y]} \end{aligned}$$
- Suppose $F = G \& G'$.

$$\begin{aligned} \llbracket G \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]}; \llbracket G' \rrbracket^{V[x \mapsto X]} \subseteq \llbracket G' \rrbracket^{V[x \mapsto Y]} && \text{[By IH]} \\ \Rightarrow \llbracket G \rrbracket^{V[x \mapsto X]} \cap \llbracket G' \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]} \cap \llbracket G' \rrbracket^{V[x \mapsto Y]} \\ \Rightarrow \llbracket F \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket F \rrbracket^{V[x \mapsto Y]} \end{aligned}$$
- Suppose $F = G \oplus G'$.

$$\begin{aligned} \llbracket G \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]}; \llbracket G' \rrbracket^{V[x \mapsto X]} \subseteq \llbracket G' \rrbracket^{V[x \mapsto Y]} && \text{[By IH]} \\ \Rightarrow \llbracket G \rrbracket^{V[x \mapsto X]} \cup \llbracket G' \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket G \rrbracket^{V[x \mapsto Y]} \cup \llbracket G' \rrbracket^{V[x \mapsto Y]} \\ \Rightarrow (\llbracket G \rrbracket^{V[x \mapsto X]} \cup \llbracket G' \rrbracket^{V[x \mapsto X]})^{\perp\perp} &\subseteq (\llbracket G \rrbracket^{V[x \mapsto Y]} \cup \llbracket G' \rrbracket^{V[x \mapsto Y]})^{\perp\perp} && \text{[By Proposition 4]} \\ \Rightarrow \llbracket F \rrbracket^{V[x \mapsto X]} &\subseteq \llbracket F \rrbracket^{V[x \mapsto Y]} \end{aligned}$$
- Suppose $F = \mu y.G$. Observe that $y \neq x$ since we assumed that x is not a bound variable in F . Now by hypothesis, for any fact Z , $\llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]} \subseteq \llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]}$. Therefore for every Z such that $\llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]} \subseteq Z$ we have $\llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]} \subseteq Z$. Therefore, $\{Z \mid \llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]} \subseteq Z\} \subseteq \{Z \mid \llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]} \subseteq Z\}$. Hence, $\bigcap_{\llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]} \subseteq Z} \{Z\} \subseteq \bigcap_{\llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]} \subseteq Z} \{Z\}$. We conclude $\llbracket F \rrbracket^{V[x \mapsto X]} \subseteq \llbracket F \rrbracket^{V[x \mapsto Y]}$.
- Suppose $F = \nu y.G$. As before we comment that $y \neq x$ and therefore by hypothesis, for any fact Z , $\llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]} \subseteq \llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]}$. Therefore for every Z such that $Z \subseteq \llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]}$ we have $Z \subseteq \llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]}$. Therefore, $\{Z \mid Z \subseteq \llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]}\} \subseteq \{Z \mid Z \subseteq \llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]}\}$. Hence, $\bigcup_{Z \subseteq \llbracket G \rrbracket^{V[x \mapsto X, y \mapsto Z]}} \{Z\} \subseteq \bigcup_{Z \subseteq \llbracket G \rrbracket^{V[x \mapsto Y, y \mapsto Z]}} \{Z\}$. Applying Proposition 4 twice, we conclude $\llbracket F \rrbracket^{V[x \mapsto X]} \subseteq \llbracket F \rrbracket^{V[x \mapsto Y]}$. \blacktriangleleft

B.1.2 Proof of Theorem 28

Proof. We show it for μf . First of all, $\{X \mid f(X) \subseteq X\}$ is non-empty since $M^\perp \in \mathcal{D}$. First, we show that it is indeed a fixpoint. Observe that $\mu f \subseteq X$ for any $X \in \mathcal{D}$ which is a pre-fixpoint of f . By Lemma 27, one can apply f on both sides. So $f(\mu f) \subseteq f(X)$ for all $X \in \mathcal{D}$ satisfying $f(X) \subseteq X$ and therefore $f(\mu f) \subseteq \bigcap_{X \in \mathcal{D}} \{X \mid f(X) \subseteq X\} = \mu f$. So μf is a prefixpoint.

But then, since $\mu f \in \mathcal{D}$, and thanks to the closure properties of \mathcal{D} , so is $f(\mu f)$. By monotonicity of f , one gets that $f(f(\mu f)) \subseteq f(\mu f)$, ensuring that $f(\mu f)$ is a prefixpoint of f . But μf is the least prefixpoint; so, we conclude that $\mu f \subseteq f(\mu f)$. Therefore, $\mu f = f(\mu f)$. Finally recall $\mu f \subseteq X$ for any pre-fixpoint X in \mathcal{D} , so it is the least fixpoint in \mathcal{D} . \blacktriangleleft

B.1.3 Proof of Lemma 29

Proof. By induction on F . The base case is when F is an atom, a variable or a unit.

- Suppose $F = a \in \mathcal{A}$. Then, $\llbracket F^\perp \rrbracket^{V^\perp} = V^\perp(a^\perp) = V(a^\perp) = V(a)^\perp = (\llbracket F \rrbracket^V)^\perp$.
- Suppose $F = x \in \mathcal{V}$. Then, $\llbracket F^\perp \rrbracket^{V^\perp} = V^\perp(x^\perp) = V^\perp(x) = V(x)^\perp = (\llbracket F \rrbracket^V)^\perp$.
- The case for the units is easy.

There are several subcases for the induction case.

- Suppose $F = G \otimes G'$.

$$\begin{aligned}
\llbracket (G \otimes G')^\perp \rrbracket^{V^\perp} &= \llbracket G^\perp \wp G'^\perp \rrbracket^{V^\perp} \\
&= ((\llbracket G^\perp \rrbracket^{V^\perp})^\perp (\llbracket G'^\perp \rrbracket^{V^\perp})^\perp)^\perp \\
&= (\llbracket G^{\perp\perp} \rrbracket^V \llbracket G'^{\perp\perp} \rrbracket^V)^\perp && \text{[By IH]} \\
&= (\llbracket G \rrbracket^V \llbracket G' \rrbracket^V)^\perp \\
&= (\llbracket G \rrbracket^V \llbracket G' \rrbracket^V)^{\perp\perp\perp} && \text{[By Proposition 4]} \\
&= (\llbracket (G \otimes G') \rrbracket^V)^\perp
\end{aligned}$$

Negating both sides we have, $(\llbracket (G \otimes G')^\perp \rrbracket^{V^\perp})^\perp = (\llbracket (G \otimes G') \rrbracket^V)^{\perp\perp}$. Hence, $(\llbracket G^\perp \wp G'^\perp \rrbracket^{V^\perp})^\perp = \llbracket (G^\perp \wp G'^\perp)^\perp \rrbracket^V$. This takes care of the case when the outermost connective of F is a \wp .

- Suppose $F = G' \oplus G'$.

$$\begin{aligned}
\llbracket (G' \oplus G')^\perp \rrbracket^{V^\perp} &= \llbracket G^\perp \& G'^\perp \rrbracket^{V^\perp} \\
&= \llbracket G^\perp \rrbracket^{V^\perp} \cap \llbracket G'^\perp \rrbracket^{V^\perp} \\
&= (\llbracket G \rrbracket^V)^\perp \cap (\llbracket G' \rrbracket^V)^\perp && \text{[By IH]} \\
&= (\llbracket G \rrbracket^V \cup \llbracket G' \rrbracket^V)^\perp \\
&= (\llbracket G \rrbracket^V \cup \llbracket G' \rrbracket^V)^{\perp\perp\perp} && \text{[By Proposition 4]} \\
&= (\llbracket G \oplus G' \rrbracket^V)^\perp
\end{aligned}$$

As in the previous case, negating both sides, we derive the case when the outermost connective of F is a $\&$.

- Suppose $F = \mu x.G$.

$$\begin{aligned}
\llbracket (\mu x.G)^\perp \rrbracket^{V^\perp} &= \llbracket \nu x.G^\perp \rrbracket^{V^\perp} \\
&= \left(\bigcup_{X \in \mathcal{D}} \{X \mid X \subseteq \llbracket G^\perp \rrbracket^{V^\perp[x \rightarrow X]}\} \right)^{\perp\perp} \\
&= \left(\bigcup_{X \in \mathcal{D}} \{X \mid X \subseteq \llbracket G^\perp \rrbracket^{(V[x \rightarrow X^\perp])^\perp}\} \right)^{\perp\perp} \\
&= \left(\bigcup_{X \in \mathcal{D}} \{X \mid X \subseteq (\llbracket G \rrbracket^{V[x \rightarrow X^\perp]})^\perp\} \right)^{\perp\perp} && \text{[By IH]} \\
&= \left(\bigcap_{X \in \mathcal{D}} \{X^\perp \mid X \subseteq (\llbracket G \rrbracket^{V[x \rightarrow X^\perp]})^\perp\} \right)^\perp && \text{[By Proposition 4]} \\
&= \left(\bigcap_{X \in \mathcal{D}} \{X^\perp \mid \llbracket G \rrbracket^{V[x \rightarrow X^\perp]} \subseteq X^\perp\} \right)^\perp && \text{[By Proposition 4]} \\
&= \left(\bigcap_{X \in \mathcal{D}} \{X \mid \llbracket G \rrbracket^{V[x \rightarrow X]} \subseteq X\} \right)^\perp && \text{[Closure property of } \mathcal{D} \text{]} \\
&= (\llbracket \mu x.G \rrbracket^V)^\perp
\end{aligned}$$

As in the previous case, negating both sides, we derive the case when the outermost operator of F is a ν . ◀

B.2 Proofs of Section 4

B.2.1 Proof of Lemma 45

Proof. We induct on $|\text{rk}(F)|$. The base case is when $|\text{rk}(F)| = 1$. Since $x \in \text{fv}(F)$, F cannot be an atom or a unit. Therefore, $F = x$. Plugging $\sigma = \langle \rangle$, we are done. The induction case has several subcases.

- Suppose $F = G \odot G'$ where $\odot \in \{\otimes, \wp, \&, \oplus\}$. We have two cases. Either $x \in \text{fv}(G) \cap \text{fv}(G')$ or x is free in only one of them. Note that $\text{rk}(G), \text{rk}(G') \leq \text{rk}(F)$. Therefore if x is free in them, the induction hypothesis can be fired. In the first case, we have $\text{rk}(G(\xi/x)) = \text{rk}(\xi) * \sigma_1$ and $\text{rk}(G'(\xi/x)) = \text{rk}(\xi) * \sigma_2$. Therefore,

$$\begin{aligned} \text{rk}(F(\xi/x)) &= \text{rk}(G(\xi/x) \odot \text{rk}(G'(\xi/x))) \\ &= (\text{rk}(G(\xi/x)) \sqcup \text{rk}(G'(\xi/x))) * \langle 0 \rangle \\ &= ((\text{rk}(\xi) * \sigma_1) \sqcup (\text{rk}(\xi) * \sigma_2)) * \langle 0 \rangle \\ &= \text{rk}(\xi) * (\sigma_1 \sqcup \sigma_2) * \langle 0 \rangle \end{aligned}$$

Therefore by plugging $\sigma = (\sigma_1 \sqcup \sigma_2) * \langle 0 \rangle$, we are done. In the other case, wlog assume $x \notin \text{fv}(G')$. Therefore, we have $G'(\xi/x) = G'$. Firing the induction hypothesis for G , we have $\text{rk}(G(\xi/x)) = \text{rk}(\xi) * \sigma_1$ as before. Therefore,

$$\begin{aligned} \text{rk}(F(\xi/x)) &= \text{rk}(G(\xi/x) \odot \text{rk}(G'(\xi/x))) \\ &= (\text{rk}(G(\xi/x)) \sqcup \text{rk}(G')) * \langle 0 \rangle \\ &= ((\text{rk}(\xi) * \sigma_1) \sqcup \text{rk}(G')) * \langle 0 \rangle \\ &= \text{rk}(\xi) * \sigma_1 * \langle 0 \rangle \quad [\text{Since } \text{rk}(G') \leq \text{rk}(\xi)] \end{aligned}$$

Therefore by plugging $\sigma = \sigma_1 * \langle 0 \rangle$, we are done.

- Suppose $F = \eta^\beta y.G$ where $\eta \in \{\mu, \nu\}$, $\beta < \omega$, and $y \neq x$. Clearly, $x \in \text{fv}(G)$ and $\text{rk}(G) \leq \text{rk}(F)$. Therefore, by hypothesis, $\text{rk}(G(\xi/x)) = \text{rk}(\xi) * \sigma'$.

$$\begin{aligned} \text{rk}(F(\xi/x)) &= \text{rk}(\eta^\beta y.G(\xi/x)) \\ &= \text{rk}(G(\xi/x)) * \langle \beta \rangle \\ &= (\text{rk}(\xi) * \sigma') * \langle \beta \rangle \end{aligned}$$

By plugging $\sigma = \sigma' * \langle \beta \rangle$, we are done. The case $F = \eta y.G$ goes exactly similarly. \blacktriangleleft

B.2.2 Proof of Lemma 53

Proof. We recall that $\Gamma_0 = \mu x.G, a \otimes \nu y.F$ and we will show that this sequent is not provable in μMALL_ω . Suppose there is a proof. By Corollary 52, we can assume that this proof is cut-free. Therefore, the only possibilities for the first rule are (\otimes) or one of $\{(\mu_n^\omega)\}_{n \in \omega}$. If it is the former, then the left premisses has to contain $\mu x.G$ since $\vdash a$ cannot be proved. Consequently, the right premisses is $\vdash \nu y.F$. If it were provable, so would be $\vdash \nu^n x.F$ for all $n \in \omega$. It is easy to observe that $\vdash \nu^n x.F$ is not provable for all $n > 0$.

Now suppose the first rule is a (μ_n^ω) for some $n \in \omega$. We note that the (μ^n) rule is invertible. Moreover, the (\wp) rule is also invertible. Therefore, it suffices to show that $\vdash (a^\perp)^n, \mathbf{0}, a \otimes \nu y.F$ is not provable. The only possible rule here is the (\otimes) . The only splitting of the context which renders the left premiss provable is one where the right premiss is $\vdash \overbrace{a^\perp, \dots, a^\perp}^{n-1}, \mathbf{0}, \nu y.F$. The only possible rule that can be applied here is the (ν^ω) rule. Consider any premiss other than the $\lfloor \frac{n-1}{2} \rfloor$ th premiss. Since the number of a and a^\perp are different in it, it is not provable by Example 7. \blacktriangleleft

Natural Colors of Infinite Words

Rüdiger Ehlers   

Technische Universität Clausthal, Germany

Sven Schewe   

University of Liverpool, UK

Abstract

While finite automata have minimal DFAs as a simple and natural normal form, deterministic omega-automata do not currently have anything similar. One reason for this is that a normal form for omega-regular languages has to speak about more than acceptance – for example, to have a normal form for a parity language, it should relate every infinite word to some natural color for this language. This raises the question of whether or not a concept such as a natural color of an infinite word (for a given language) exists, and, if it does, how it relates back to automata.

We define the natural color of a word purely based on an omega-regular language, and show how this natural color can be traced back from any deterministic parity automaton after two cheap and simple automaton transformations. The resulting *streamlined* automaton does not necessarily accept every word with its natural color, but it has a “co-run”, which is like a run, but can *once* move to a language equivalent state, whose color is the natural color, and no co-run with a higher color exists.

The streamlined automaton defines, for every color c , a good-for-games co-Büchi automaton that recognizes the words whose natural colors with respect to the represented language are at least c . This provides a canonical representation for every ω -regular language, because good-for-games co-Büchi automata have a canonical minimal – and cheap to obtain – representation for every co-Büchi language.

2012 ACM Subject Classification Theory of computation \rightarrow Automata over infinite objects; Theory of computation \rightarrow Linear logic; Theory of computation \rightarrow Logic and verification

Keywords and phrases parity automata, automata over infinite words, ω -regular languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.36

Funding *Rüdiger Ehlers*: This work was supported by the German Science Foundation (DFG) under Grant No. 322591867 (GUISynth).

Sven Schewe: This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement 956123 (FOCETA). It was also supported by the EPSRC through project EP/X017796/1.



Acknowledgements We thank Arved Friedemann for interesting discussions that had a positive influence on the undertaking of this work.

1 Introduction

A classical question in the theory of automata is how to define *canonical representations* of regular languages. Such a representation, typically in the form of an automaton, for a language has several advantages. For once, different canonical automata must define different languages. But reasonably defined canonical automata are also concise and normally a minimal (and thereby natural) representative of all language equivalent automata of the same type, which makes them a natural representative of the language they recognize.

Such definitions of canonicity build on – or deliver – insights into the possible structure of an automaton for a given language. For instance, canonical deterministic automata over finite words have exactly one state per (reachable) suffix language, and the Myhill-Nerode automaton minimization procedure is able to translate every deterministic automaton over finite words into its canonical form in polynomial time [6]. The concepts that underpin



© Rüdiger Ehlers and Sven Schewe;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 36; pp. 36:1–36:17



Leibniz International Proceedings in Informatics

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

insightful canonicity definitions often give rise to efficient minimization procedures, which makes it attractive to apply them in practical applications. In turn, such concepts are useful in concisely defining a language, including for practical applications like learning, model checking, or synthesis.

For regular languages over infinite words, obtaining insightful canonical forms has remained a challenge. Such languages are useful for reasoning about *reactive systems*, i.e., computational systems that continuously read from an input stream while producing an output stream. While Löding [8] gave a construction for computing canonical and minimal deterministic *weak* automata, which can encode some such languages, this automata class is very restricted in that in every strongly connected component, either all states are accepting or all states are rejecting. This means that simple languages such as “there are (in)finitely many *a*s in the word” cannot be represented by them.

After the result by Löding [8], there was, for quite a while, little progress on canonical forms for more expressive subclasses of ω -regular languages. This is partially rooted in the fact that deterministic Büchi (and co-Büchi) automata – which are among the simplest ω -automaton types and cannot even capture all ω -regular languages – have an NP-complete minimization problem [11]. This implies that, unlike in the case of languages over finite words, deterministic automata with the common state based acceptance cannot be used for defining a canonical form that is easy to compute.

Only very recently, Abu Radi and Kupferman [1] observed that, via a slight generalization from deterministic co-Büchi automata to (transition-based) good-for-games [5] co-Büchi automata, we obtain an automaton model for co-Büchi languages that permits a polynomial-time minimization procedure; this gives rise to an insightful canonical form. Transition-based good-for-games co-Büchi automata (and similarly, good-for-games parity automata with a fixed set of colors) are not more expressive than deterministic automata with the same acceptance condition [5], but they can be more concise. Interestingly, this added conciseness is what enables polynomial-time minimization and thereby efficiently computing canonical automata. In the canonical minimal automata computed using the construction by Abu Radi and Kupferman [1], non-determinism only appears along *rejecting* transitions, which connect different strongly connected components that consist only of accepting transitions. Hence, the different deterministic strongly connected components represent the different ways in which a word can be accepted and hence provide insight into the structure of the represented language.

The result by Abu Radi and Kupferman raises the question of whether this result can be extended to obtain a canonical and insightful representation for general ω -regular languages or not. Such an extension would intuitively need to use a richer type of acceptance condition than co-Büchi acceptance, as co-Büchi acceptance is too limited in expressivity. The weakest acceptance condition that offers full ω -regularity in this context is *parity acceptance*. In parity automata, a word is accepted if, and only if, the lowest *color* that occurs infinitely often along a run of the automaton is even.

We could salvage the polynomial-time canonicalization procedure for co-Büchi acceptance while using a parity-type acceptance condition by representing an ω -regular language L by a falling chain of languages $L_0 \supset L_1 \supset L_2 \dots \supset L_c$ such that L_0 is the universal language and each language L_i is a co-Büchi language. A word is then accepted by the *chain* of languages if, and only if, the highest i such that the word is in L_i is even. As all of these languages are co-Büchi languages, we can represent each of them by their canonical minimal good-for-games automaton such that, together, these automata are a canonical representation of L .

The crucial piece that is currently missing in the literature to obtain such a canonical representation of an arbitrary ω -regular language, however, is which word should be in which language L_i . Omega-regular languages can be decomposed into such chains in different ways,

and for the overall chain to be a canonical representation of the language, we need to fix a way for decomposing the language L into co-Büchi languages L_i . In other words, we are missing a definition of the *natural* color of a word that defines the highest index i such that the word is in L_i . This natural color depends on the overall language to be represented, as this color reflects where in the decomposition of a given language the word resides.

For a useful canonicity definition, we need the allocation of words to the individual L_i for a given ω -regular language L to have several properties:

1. the definition should be based on the language L alone, and be independent of the syntactic structure of any representation of it (such as some parity automaton that recognizes L),
2. the definition should be easy to compute for a given word and a given representation of L (such as a deterministic parity automaton), and
3. starting from an automaton representation of L , the sizes of co-Büchi automata for the languages L_i should be *small*, ideally not bigger than the size of an automaton for L .

In this paper, we provide a definition of a *natural* color of an infinite word for a given ω -regular language that has these properties. Our definition distills the idea that, in a parity automaton, only the lowest color visited infinitely often along a run matters, into a concept that can be defined on languages alone, without referring to a specific automaton. We then use this for introducing a canonical representation of arbitrary ω -regular languages as a chain of co-Büchi languages. While the definition of the natural color of a word (for a given language) is the main technical contribution of this paper, its study is motivated by what it can be used for, namely for establishing a canonical representation for ω -regular languages, which is the conceptual contribution of this paper.

We show that our particular definition of the natural color of a word (for a given language) has the property that every deterministic parity automaton can be translated into a form from which the natural color of a word can easily be read off. This works in two steps: We first simplify an automaton by ordering its strongly connected components (using an order that respects reachability) and bending all transitions to language equivalent states in the maximal component they reside in (besides removing unreachable and unproductive states). In a second step, we construct a so-called *streamlined* form of a parity automaton that retains the transition structure. Both transformations are tractable.

From a streamlined automaton, we can furthermore obtain, again in polynomial time, good-for-games co-Büchi automata for all languages L_i . They are no larger than the original streamlined parity automaton, and therefore no larger than the deterministic parity automaton we started with. Moreover, they can subsequently be minimized [1] to obtain a canonical representation of a given ω -regular language. This minimization can also yield an exponential advantage over a representation as deterministic co-Büchi automata [7].

As a consequence, with our definition, one can obtain, in polynomial time, a canonical representation of the language of a deterministic parity automaton. While this representation is not a single automaton, deviating from deterministic branching was necessary in order to avoid the NP-hardness of minimizing deterministic parity automata. Furthermore, it was shown that good-for-games parity (incl. Büchi and co-Büchi) automata are also NP-hard to minimize [12] when using state-based acceptance, while the complexity of minimizing Büchi (and, more generally, parity) good-for-games automata with transition-based acceptance is still open, so a further generalization had to be made. By choosing a sequence of good-for-games transition-based co-Büchi automata as this generalization, we avoid introducing a more complex automaton type at the cost of having multiple automata.

While it is possible to define other variants of what the natural color of a word (for a given language) could be, our definition has the advantage that it coincides with the color of a word of *some* parity automaton for a given language while permitting a translation from a deterministic parity word automaton to a canonical representation of its language in polynomial time.

2 Preliminaries

Given a set S , we denote the set of finite sequences (words) of elements in S as S^* and the set of infinite sequences of elements in S as S^ω . Sets of words are also called *languages* (over some alphabet). We only consider finite alphabets. We denote the set of natural numbers including 0 by \mathbb{N} . Given a language $L \subseteq \Sigma^\omega$ and a finite word $w \in \Sigma^*$, we define the suffix language of L over w as $\mathcal{L}^{\text{suffix}}(L, w) = \{w' \in \Sigma^\omega \mid ww' \in L\}$.

We define *parity automata (with transition-based acceptance)* as tuples $\mathcal{A} = (Q, \Sigma, \delta, Q_0)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q \times \mathbb{N}$ is a transition relation, and Q_0 is the set of initial states. We say that a word $w = w_0w_1 \dots \in \Sigma^\omega$ induces a run $\pi = q_0q_1 \dots$ of the automaton with the corresponding color sequence $\rho = \rho_0\rho_1 \dots \in \mathbb{N}^*$ if $q_0 \in Q_0$ and, for every $j \in \mathbb{N}$, we have $(q_j, w_j, q_{j+1}, \rho_j) \in \delta$. We say that w is *accepted* by \mathcal{A} if there exists a run ρ for the word on which the lowest color that occurs infinitely often along ρ is even. For the remainder of this paper, all automata considered employ transition-based acceptance whenever not stated otherwise.

We say that \mathcal{A} is *deterministic* if, for every state $q \in Q$ and $x \in \Sigma$, we have exactly one element of the form $(q, x, q', c) \in \delta$, and Q_0 is a singleton set. We henceforth use q_0 as tuple element for the initial state for deterministic automata. In deterministic automata, every word induces a unique run/color sequence combination. This also allows us to define, by slight abuse of notation, for each state q and word $w \in \Sigma^*$, $\delta(q, w)$ to be the unique state reached from q after reading w . We refer to the smallest color that occurs infinitely often in the color sequence corresponding to a run for w as the *color of w in \mathcal{A}* . We also call it the *dominating color* among the ones occurring infinitely often in the color sequence for w .

The set of words accepted by \mathcal{A} is called its language, also denoted by $\mathcal{L}(\mathcal{A})$. We say that \mathcal{A} is a *co-Büchi automaton* if only the colors 1 and 2 occur along transitions. Transitions with color 1 and 2 are also called *rejecting* and *accepting* transitions, respectively. The automaton \mathcal{A}_q with $q \in Q$ denotes a variant of \mathcal{A} for which the initial state has been replaced by q . We say that two states $q, q' \in Q$ of a deterministic parity automaton (DPA) \mathcal{A} are *equivalent*, denoted by $q \sim_{\mathcal{A}} q'$, if, and only if, they have the same language $\mathcal{L}(\mathcal{A}_q) = \mathcal{L}(\mathcal{A}_{q'})$.

We say that an automaton \mathcal{A} is *good-for-games* if there exists a strategy function [1] $f : \Sigma^* \rightarrow Q \times \mathbb{N}$ such that for each word $w = w_0w_1 \dots \in \Sigma^\omega$ in the language of \mathcal{A} , there exists an accepting run $\pi = q_0q_1 \dots \in Q^\omega$ with corresponding color sequence $\rho = \rho_0\rho_1 \dots \in \mathbb{N}^\omega$ for it such that for all $j \in \mathbb{N}$, we have $(q_{j+1}, \rho_j) = f(w_0 \dots w_j)$. Note that such a run is unique for each w .

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0)$, we say that a tuple $(\tilde{Q}, \tilde{\delta})$ with $\tilde{Q} \subseteq Q$ and $\tilde{\delta} \subseteq \delta \cap \tilde{Q} \times \Sigma \times \tilde{Q} \times \mathbb{N}$ is a *strongly connected component (SCC)* if, for each $q, q' \in \tilde{Q}$, we have that there exists a sequence of states q_1, \dots, q_n all in \tilde{Q} for some $n \in \mathbb{N}$ such that $q_1 = q$, $q_n = q'$, and for every $1 \leq j < n$, there exist $x \in \Sigma$ and $c \in \mathbb{N}$ such that $(q_j, x, q_{j+1}, c) \in \tilde{\delta}$. We say that $(\tilde{Q}, \tilde{\delta})$ is a *maximal SCC* if no states and transitions can be added without losing the property that the (resulting) tuple is an SCC. Transitions that can only be taken once in a run (starting from any state) are called *transient*; they connect different SCCs. We also say that a state is *transient* if it can only be visited once along a run. For some co-Büchi

automaton \mathcal{A} , we say that some SCC $(\tilde{Q}, \tilde{\delta})$ of \mathcal{A} is an *accepting SCC* if $\tilde{\delta} \subseteq \tilde{Q} \times \Sigma \times \tilde{Q} \times \{2\}$. We furthermore call $(\tilde{Q}, \tilde{\delta})$ a *maximal accepting SCC* if $(\tilde{Q}, \tilde{\delta})$ cannot be strictly extended with further states or accepting transitions without losing the property that it is an accepting SCC.

3 Towards A Canonical Language Representation

The core definition we provide in this paper, namely the *natural color of a word*, lifts the idea of parity acceptance from deterministic automata to languages. Such a natural color will always be defined with respect to a given language, but for the brevity of presentation, we will not always mention this language henceforth.

Since, at the level of languages, there are no colors of transitions that can be referred to, the definition of the natural color of a word needs to capture the idea of colors in a way that does not employ the colors of transitions.

In this section, we make some observations on why some languages need a certain number of colors in a deterministic parity automaton, and identify ways in which we can abstract from the automaton representation along the way. We then distil the observations to define the natural color of a word in the next section.

Niwiński and Walukiewicz [10] have given a polynomial-time algorithm to minimize the number of different colors in a deterministic parity automaton. While their algorithm targets parity automata in which states – rather than transitions – are labeled with colors, it is not difficult to extend it to transition-based acceptance.

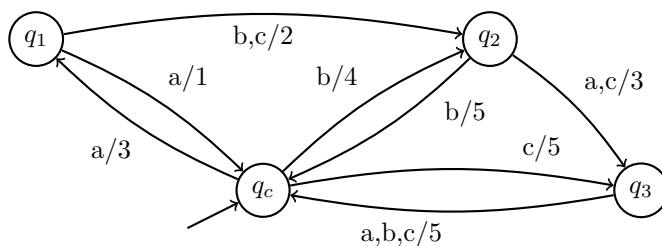
The core idea used in their algorithm is that, in order for a deterministic parity automaton that recognizes a language to need at least n colors, there needs to exist a so-called *flower* with at least n colors. Such a flower is defined to satisfy two properties. It firstly has a sequence of colors $c_1 < c_2 < \dots < c_n$ such that every two successive colors in the sequence alternate by whether they are even or odd. Secondly, there exists a center state such that, for each color c_i , there exist paths from the center state back to itself such that the dominating color occurring along the transitions along the path is c_i . Following the terminology of Niwiński and Walukiewicz, we refer to such paths as *flower loops*. Figure 1 shows an example parity automaton that contains such a flower over the colors 1, 2, 3, 4, and 5.

Niwiński and Walukiewicz have shown that no deterministic parity automaton with fewer than n colors can encode a language that admits a flower with n colors [10]. This is because a flower defines a hierarchy over words that are, alternately, accepted or rejected by an automaton, and the n different colors are needed to detect on which level in the hierarchy a word is located.

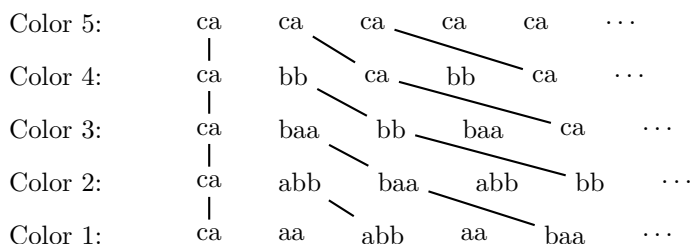
Figure 2 shows such a hierarchy of words for the parity automaton from Figure 1. They all have in common that the state q_c in the center of the flower is visited infinitely often in a run of the automaton for the respective word.

The first word, $(ca)^\omega$, leads to only transitions with color 5 being taken, so the color of the word is 5, and the word is rejected.

The second word is built by injecting bb strings at positions of the word at which the respective run is in the center state q_c . Note that bb causes transitions $q_c \xrightarrow{b} q_2 \xrightarrow{b} q_c$ in the run for the second word, so that the added string causes an excursion in the run that leads back to the same state. In this way, the run of the second word can be obtained from the run for the first word by adding elements at the positions in which a finite substring is inserted into the word. Because in the run for the second word, color 4 is visited infinitely often, this becomes the color of the modified word.



■ **Figure 1** A flower in a parity automaton. The flower loops are $q_c \xrightarrow{a} q_1 \xrightarrow{a} q_c$ for color 1, $q_c \xrightarrow{a} q_1 \xrightarrow{c} q_2 \xrightarrow{b} q_c$ for color 2, $q_c \xrightarrow{b} q_2 \xrightarrow{a} q_3 \xrightarrow{c} q_c$ for color 3, $q_c \xrightarrow{b} q_2 \xrightarrow{b} q_c$ for color 4, and $q_c \xrightarrow{c} q_3 \xrightarrow{c} q_c$ for color 5.



■ **Figure 2** An example hierarchy of words for the parity automaton from Figure 1, used in an example in Section 3. The words from the lower colors are obtained from those of higher colors by inserting language invariant words, which are flower loops in the given example. The lines show where the flower loops from the words with higher color are in the words with lower color.

The other words are built according to the same idea: By injecting finite words leading from q_c back to q_c (while taking a transition with a lower color) into the word at positions in the word on which a run for the word is at state q_c anyway, we obtain a new word that is accepted by the automaton if, and only if, the old word is rejected.

The most significant color (here: 1) now has the special property that inserting more loops does not change whether or not the word is accepted, as the most significant color of the automaton is already visited along the run. We can formalize this as follows: Let $w = w_0w_1 \dots$ be a word and $\pi = q_0q_1 \dots$ be a run of the automaton over w . We say that a finite word w' is a state-invariant injection at a position $i \in \mathbb{N}$ in the word if $\delta(q_i, w') = q_i$. We can characterize the words w that are recognized with the most significant color in a strongly connected component (SCC) of the parity automaton as those for which every word \tilde{w} that results from an infinite sequence of injections of state-invariant words into w is accepted by the automaton if, and only if, w is accepted by the automaton. Note that the restriction to strongly connected components is necessary, as other parts of a deterministic parity automaton may employ more colors, and hence this property holding for a word whose run ends in some SCC does not exclude that some other automaton part uses more colors, including a lower one.

3.1 The Case of the Most Significant Color

Let us now distill the definition of acceptance with the most significant color to the language case. In the resulting definition, the restriction to a single SCC will also be lifted.

The hierarchy of words from Figure 2 refers to the states of a given deterministic parity automaton: Finite words can only be inserted at places in which the added finite words loop from the state in which the run of the automaton is at the insertion place, back to

the same state. This idea can be lifted by replacing the notion of state by *suffix language invariance*. We say that inserting a finite word $u \in \Sigma^*$ at position $i \in \mathbb{N}$ in a word $w = w_0w_1 \dots \in \Sigma^\omega$ is a suffix language invariant injection into w for a language $L \subseteq \Sigma^\omega$ if $\mathcal{L}^{\text{suffix}}(L, w \dots w_i) = \mathcal{L}^{\text{suffix}}(L, w \dots w_i u)$.

In this definition, the suffix languages take the role of the flower center states, and they have the nice property that they are independent of an automaton representation of the language.

Injecting any finite number of loops from a state back to itself into a run of a parity automaton does not change the color with which the respective word is accepted. It makes sense to expect for the definition of the natural color of a word that similarly, any finite number of suffix language invariant injections should not change the color of a word.

With this in mind, we can try to liberate the definition regarding which words should be recognized with the most significant color from any reference to a particular automaton: they are those words for which an infinite number of suffix language invariant word injections does not change whether or not a word is accepted. It is, however, necessary to carefully define where exactly in a word these injections can be made.

To see this, consider the case of the language “there are infinitely often two *as* in a row”. It can be recognized by a deterministic *Büchi* automaton, i.e., a deterministic parity automaton with colors 0 and 1. Since 0 is the most significant color, all words in the language need to have this natural color. This language has only a single suffix language, namely itself. If we would require for a word to be of natural color 0 that *all* infinite sequences of suffix language invariant word injections result in an accepted word, then *no* word would be accepted with this natural color: This is because this would include injecting a *b* as every second letter in a word. Consequently, not all injection sequences need to be tolerated for a word to be of natural color 0. But it also does not suffice if only *some* injection is tolerated: In an extreme case, that includes injecting the empty word everywhere, which never affects acceptance.

A compromise between these two extremes is to use different quantifiers for the points of injection and the words being injected. We declare those words to have a lowest natural color, for which *there exists* an infinite sequence of points, at which suffix invariant words can be injected, such that, *for all* insertions of sequences of suffix invariant words at these points, the resulting word is accepted if, and only if, the original word was. While this solves the problem from the short example in the paragraph above, it is not trivially clear whether or not other problems remain when using this definition. The correctness of the construction from the next section, however, shows that this is precisely the definition we need.

3.2 Generalizing to All Colors

To generalize the idea of the natural color of a word from the most significant color to the general case, we can follow an inductive argument and – in a sense – peel the language off, layer by layer. We look at the colors $c \in \mathbb{N}$ in ascending order and define, for each color $c + 1$, which words are natural for this color, under the assumption that we have such a definition for colors up to c . To do so, we can marry an inductive definition of what constitutes the color of a word in a deterministic parity automaton with the automaton-agnostic definition of the natural color of a word for the most significant color from above.

We start with revisiting an inductive definition of the color of a word in a deterministic parity automaton. We can characterize the words accepted with color 0 to be those along whose runs transitions with color 0 are taken infinitely often. For colors $c > 0$, we can define that a word is accepted with color c if transitions with color c are taken infinitely often for its run, *and* the word is not accepted with a color smaller than c . The nice property of this rather indirect definition is that it only refers to colors already defined and a single additional color.

In an orthogonal composition of this idea for an inductive definition with the central idea from the previous subsection, we can allocate the color c as the natural color of a word w if there exists an infinite set of indices such that, for every sequence of suffix invariant strings inserted at these indices, we have that the resulting word \tilde{w}

- *either* has a natural color smaller than c ,
- *or* it does not, and \tilde{w} is in the language if, and only if, w is in the language.

Thus, we only require that inserting the words makes no difference regarding acceptance where the resulting words are not of a smaller natural color.

This definition has the nice property that, by induction, for every color, the natural colors of words are uniquely defined purely by the language of the word, without reference to an automaton representation. The concrete definition given in the next section, however, makes the words for colors $c + i$ (for $i \geq 0$) contained in the words for a color c , so we define the natural color to be the *minimal* color to which this definition is applicable.

While it is, again, not trivially clear that this idea captures all necessary aspects of the natural color of a word, our results in the following section show that this is the case.

4 The Natural Color of a Word

In this section, we define our notion of the natural color of a word (with respect to a given language) based on the observations from the previous section. Based on this definition, we show how a sequence of co-Büchi automata can be obtained from the deterministic parity automaton such that, after the minimization of the co-Büchi automata, the sequence is a canonical representation of the language. This construction has two steps, namely first *streamlining* the deterministic parity automaton, and then extracting the co-Büchi automata from it. Using this sequence of automata as representation for the natural colors of words (with respect to the represented language), we finally show that the natural colors of words can be read off from the streamlined deterministic parity automaton directly, which shows that the natural colors of words can be read off from *any* deterministic parity automaton after preprocessing it.

We start with defining the colors in which a word is “at home” for a given language. The natural color of a word is then the minimal color in which a word is at home. First, we repeat some concepts from the previous section to make Definition 1 below self-contained. We say that some finite word $u \in \Sigma^*$ is *suffix language invariant* for a language L after a finite word $w \in \Sigma^*$ if we have $\mathcal{L}^{\text{suffix}}(L, w) = \mathcal{L}^{\text{suffix}}(L, wu)$.

Let a word $w = w_0 w_1 \dots \in \Sigma^\omega$ over an alphabet Σ be given. We say that some word $w' \in \Sigma^\omega$ is the result of a suffix language invariant injection of a sequence of words u_0, u_1, \dots at positions $J = \{i_0, i_1, \dots\}$ with $i_0 < i_1 < \dots$ in w if, and only if, $w' = w_0 w_1 \dots w_{i_0} u_0 w_{i_0+1} \dots w_{i_1} u_1 w_{i_1+1} \dots w_{i_2} u_2 \dots$ and for each $j \in \mathbb{N}$, we have that u_j is suffix language invariant for $w_0 \dots w_{i_j}$.

► **Definition 1.** *For every even/odd i , we say that a word $w = w_0 w_1 \dots \in \Sigma^\omega$ is at home in color $i \in \mathbb{N}$ for some language L if there exists an infinite subset $J \subseteq \mathbb{N}$ such that, for every possible sequence of finite words u_0, u_1, \dots , if a word w' is the result of a suffix language invariant injection of u_0, u_1, \dots at positions J , then we have that*

- *w' is already at home in a color strictly smaller than i , or*
- *both w and w' are in L and i is even, or both w and w' are not in L and i is odd.*

Note that the first case in the preceding definition cannot apply for color $i = 0$ (as there is no smaller color), so only the second case is of relevance for $i = 0$.

We call the minimal natural number that a word w is at home in the *natural color of w* (for a given language L).

As a small remark, this definition could also be given in a variant where the lowest color a word can be at home in is 1, which swaps the order in which we check for “ i is even” and “ i is odd”. This affects the number of different natural colors that the words can have (for a given language) by at most 1. All results given henceforth also work with odd and even colors swapped.

The color of a run of a DPA is not necessarily the natural color of the word defined above (for the language of the DPA). We will, however, show how a deterministic parity automaton can be used to define a family of good-for-games co-Büchi automata that can determine the natural color of a word.

4.1 Streamlining DPAs

The first concept to use is the concept of a *structured parity automaton*: We call a DPA *structured* if (1) all states of \mathcal{A} are reachable and (2) if two states q and q' are equivalent, then they are in the same maximal SCC.

Turning a given DPA into an equivalent structured DPA is cheap.¹ First, non-reachable states are removed. Then, an arbitrary minimal preorder that preserves reachability among the maximal SCCs of the DPA is defined. Two states are equivalent according to this preorder if, and only if, they are reachable from each other (i.e., they are in the same maximal SCC). Apart from this, the preorder follows the reachability relation in that, if a state q is reachable from a state q' , then $q \geq q'$.

A transition to a state q with a language equivalent state q' such that $q' > q$ is then re-directed to some language equivalent state q'' that is maximal according to this preorder.

As the position in this preorder can only grow along every run, there are only finitely many re-directions taken on every run. The language of the automaton is not changed by this operation.

Finally, the states that become unreachable by rerouting the transitions are removed again to make the resulting automaton structured.

► **Definition 2.** Let $\mathcal{A} = (Q, \Sigma, \delta, q_0)$ be a structured DPA. We define its *streamlined version* to be the outcome of the following streamlining process, in which the structure of \mathcal{A} is not changed, but a new color is assigned to each transition.

We first produce a copy of the structure of \mathcal{A} , creating a fresh coloring graph $\mathcal{G} = (Q, \Sigma, \delta)$ (which is called “coloring graph” because it is used for determining the new colors in \mathcal{A} ; \mathcal{G} itself does not have colors). We will then successively remove states and transitions from \mathcal{G} (not from \mathcal{A}), while assigning the transitions new colors in \mathcal{A} .

Starting with $i = 0$, we do the following until \mathcal{G} is empty.

1. We first partition \mathcal{G} into maximal SCCs.
2. We then identify all transient transitions in \mathcal{G} . We change their colors in \mathcal{A} to i , and then remove the transient states and transitions from \mathcal{G} .
3. We check if in any maximal SCC of \mathcal{G} , the least color of the transitions (in \mathcal{A}) between states in the SCC has the same parity as i .
 - If such transitions are found, we first change their colors in \mathcal{A} to i , remove them from \mathcal{G} , and then return to (1) without incrementing i .
 - If there are no such transitions, we increment i and go back to (1).

These steps are repeated until \mathcal{G} is empty.

¹ The procedure can, for instance, be found in [11]; while it is only described for Büchi and co-Büchi automata there, it can also be applied to parity automata, as done in [9].

The purpose of the construction is to iteratively *lower* the colors of transitions in \mathcal{A} towards the most significant one whenever that is possible without changing the language of the automaton. The *structure* of the automaton is not altered. The algorithm identifies transitions that can be recolored to a lower color i because they can only be taken finitely often along runs that do not have a dominating color lower than or equal to i anyway. Also, the algorithm identifies transitions that can be safely recolored to i because, while changing the color to i changes the dominating color of some runs, it never changes the parity of the dominating color.

The streamlining construction above is a variant of an algorithm by Carton and Maceiras [4] to relabel the colors of states in a deterministic parity automaton with state-based acceptance. The construction has been adapted to the case of transition-based acceptance and gives rise to a more concise correctness argument stated next. It retains the structured automaton's property that, for each set of language equivalent states, all states in the set can be found in the same maximal SCC.

► **Lemma 3.** *The streamlining process terminates and does not change the language of \mathcal{A} .*

Proof. We first observe that, by a simple inductive argument, before i is incremented, all transitions with color $\leq i$ have been removed from \mathcal{G} .

As induction basis, for $i = 0$, every transition with color 0 that remains after step (2) is the minimal color in their maximal SCC, and therefore removed in step (3).

For the induction step, after incrementing the counter to $i + 1$, all transitions with color $\leq i$ have been removed by the induction hypothesis, such that all remaining transitions with color $i + 1$ must either be transient (and removed in step (2)), or of minimal color in a maximal SCC (and are then removed in step 3).

The algorithm always terminates because the color of a transition is only ever changed once (as the transition is removed from \mathcal{G} whenever their color in \mathcal{A} is changed). When no color is (re)assigned in \mathcal{A} and removed from \mathcal{G} in an iteration, i is increased. Finally, once i exceeds the number of colors of \mathcal{A} , the graph \mathcal{G} must be empty, leading to termination.

The induction argument above also establishes that the color of each transition can only be *reduced* by this construction, but to no color lower than any (new) color of the edges that have previously been removed from \mathcal{G} .

This observation is the basis for establishing language equivalence. We establish this language equivalence step-wise, considering only the effect that the changes of colors initiated by step (2) or step (3) in one iteration have on the dominating color of *some* (arbitrary) run of \mathcal{A} .

For colors changed in step (2) of the algorithm, we observe that, if the respective transition t occurs infinitely often, some other transition that has previously been removed from \mathcal{G} must occur infinitely often, too. The color of any of these removed transitions is $\leq i$, such that changing the color of t to i does not change the dominating color of the run. If t however occurs only finitely often along the run in \mathcal{A} , it cannot change the dominating color of the run either.

For a transition t whose color is changed in step (3), we distinguish three cases. First, if t occurs only finitely often along the run under consideration, the color change does not influence the dominating color of the run. Second, if the run eventually remains in the same maximal SCC (in \mathcal{G}) as t was and t occurs infinitely often, then the previous color of t was the dominating color of the run, because t 's color was minimal among the colors of the SCC. But then the new dominating color is i , which has the same parity as (and is no greater than) the previous color of t . Finally, if the run does not eventually get stuck in the maximal SCC

of t in \mathcal{G} , but t occurs infinitely often, then there are infinitely many transitions passed that have been re-colored before t , and that therefore have a color $\leq i$, such that the re-coloring of t does not change the dominating color of the run. \blacktriangleleft

4.2 From Streamlined DPAs to Color-Recognising GCAs

We will not relate the colors of the runs in streamlined automata directly to the natural color of a word, but use them to define good-for-games co-Büchi automata (GCAs) that do so. These GCAs are easy to obtain from \mathcal{A} .

► **Definition 4.** Let $\mathcal{A} = (Q, \Sigma, \delta, q_0)$ be a streamlined automaton and i be a color that occurs in \mathcal{A} , then $\mathcal{A}_i = (Q, \Sigma, \delta_i, q_0)$ is the automaton such that, for all $(q, x, q', c) \in \delta$,

- $(q, x, q', 2) \in \delta_i$ if $c \geq i$,
- $(q, x, q', 1) \in \delta_i$ if $c < i$, and
- $(q, x, q'', 1) \in \delta_i$ for all $q'' \in Q$ with $q' \sim_{\mathcal{A}} q''$ such that, for all colors c' , $(q, x, q'', c') \notin \delta$.

The co-Büchi automata are defined such that, for all colors i , \mathcal{A}_i accepts those words for which the run in \mathcal{A} has a dominating color of at least i (using transitions of the first two types in the list above only). However, \mathcal{A}_i accepts some additional words: The transitions added by the third item in the list above allow a run to “jump” to any state that is language-equivalent in \mathcal{A} (if the transition is not already part of \mathcal{A}_i by the first two items). In accepting runs, this can only happen finitely often, though.

We will show in the remainder of this subsection that, for all i , \mathcal{A}_i is a good-for-games co-Büchi automaton that accepts exactly the words that have a natural color of at least i (with respect to the language of \mathcal{A}).

The first observation that we will use to achieve this goal is that the languages of these co-Büchi automata are obviously shrinking with growing index, simply because the transitions are the same, but some of the accepting transitions become rejecting transitions (i.e., their color changes from 2 to 1).

► **Observation 5.** For $i \leq j$, $\mathcal{L}(\mathcal{A}_i) \supseteq \mathcal{L}(\mathcal{A}_j)$ holds. Also, \mathcal{A}_0 accepts all words as it only has accepting transitions (since no color is smaller than 0) while including outgoing transitions for each state/letter pair (as \mathcal{A} is deterministic).

► **Theorem 6.** For all i , \mathcal{A}_i is good-for-games.

The proof is a pretty standard proof for co-Büchi automata: it essentially says “follow the run that has longest been through accepting transitions”.

Proof. If the automaton \mathcal{A}_i has no accepting run for a word $w = w_0 w_1 w_2 \dots$, there is nothing to show.

We now assume that w has an accepting run $\pi = q_0 q_1 q_2 \dots$, where j is the first position in π such that, for all $k \geq j$, $(q_k, w_k, q_{k+1}, 2)$ are accepting transitions.

We argue that \mathcal{A}_i can accept w with the strategy to

1. follow accepting transitions where possible; note that there is at most one outgoing accepting transition for every state/letter pair, so this selection is deterministic, and
2. if no such transition is available when reading w_k , move to a state q'_{k+1} such that there exists a run prefix $\pi' = q_0 q'_1 q'_2 \dots q'_k$ for $w_0 \dots w_k$ with some $l < k + 1$ such that the transitions taken from q'_l onwards are all accepting. In particular, state q'_{k+1} is chosen for some *lowest possible* value of l among such run prefixes (the way to choose ex aequo does not matter).

36:12 Natural Colors of Infinite Words

To see why this strategy yields an accepting run, consider a total order over all possible finite run prefixes. The prefixes are ordered by their size (starting from the smallest one), but otherwise the total order is arbitrary.

Whenever the strategy can continue along an accepting transition, it does so. When it has to take a rejecting transition after having read a finite prefix w' of w , it chooses a smallest run prefix ρ' such that w' has a unique finite run $\rho'\rho''$, where ρ'' contains only accepting transitions. The run $\rho'\rho''$ ends in some state q , and our strategy is to move to this state q .

The prefix $q_0 \dots q_j$ is somewhere in this order, say at position p . Now, if there are at least p rejecting transitions taken when following the strategy, then $q_0 \dots q_j$ will eventually be tried. From this point onward, no rejecting transitions are taken anymore in the run for w . If fewer than p rejecting transitions are taken when following the strategy, the resulting run is accepting as well.

Thus, we always have a strategy that only relies on the past, and \mathcal{A}_i is good-for-games. ◀

► **Theorem 7.** \mathcal{A}_i accepts a word w if, and only if, the natural color of w for the language $\mathcal{L}(\mathcal{A})$ is at least i .

Proof.

Induction basis: For $i = 0$, every word is accepted by \mathcal{A}_0 .

Induction step: Let us assume that the property holds for all $i' < i$ for some $i > 0$.

For the induction step, we split the “if and only if” in the claim into its two directions. We first show that (*substep 1*) a word $w = w_0w_1 \dots$ with natural color at most $i - 1$ is rejected by \mathcal{A}_i , and then argue that (*substep 2*) a word rejected by \mathcal{A}_i has a natural color of at most $i - 1$. Taking both directions together and considering the remaining words (those accepted by \mathcal{A}_i rather than those rejected by \mathcal{A}_i), we obtain that a word is accepted by \mathcal{A}_i if, and only if, its natural color is at least i , which is to be proven.

Substep 1: Here, we show that a word $w = w_0w_1 \dots$ with natural color of at most $i - 1$ is rejected by \mathcal{A}_i . If the natural color of w is strictly smaller than $i - 1$, then this follows directly from the inductive hypothesis and Observation 5. For the case of the natural color of w being exactly $i - 1$, we assume for contradiction that w has a natural color of $i - 1$ and w is accepted by \mathcal{A}_i . Using the definitions for acceptance by a co-Büchi automaton and the natural color of a word, we have that

- $\pi = q_0q_1q_2 \dots$ is an accepting run of \mathcal{A}_i on w ,
- $p \in \mathbb{N}$ is a position such that, for all $j \geq p$, $(q_j, w_j, q_{j+1}, 2) \in \delta_i$ is an accepting transition in \mathcal{A}_i , and
- $J \subseteq \mathbb{N}$ is an infinite index set such that injecting suffix language invariant words at the positions in J always results in a word w' that
 - (c1) has natural color that is strictly smaller than $i - 1$ or
 - (c2) is accepted by \mathcal{A} if, and only if, $i - 1$ is even,
 where we assume w.l.o.g. $j \geq p$ for all $j \in J$.

The accepting run π has, from position p onward, only transitions that have color of at least i in \mathcal{A} . Let $p' \geq p$ be the position from which these transitions are all in the same maximal accepting SCC S in \mathcal{A}_i . By the assumption that \mathcal{A} is streamlined (which is a precondition to applying Definition 4), the maximal accepting SCC (of \mathcal{A}_i) has a transition that has a corresponding transition of color i in \mathcal{A} . To see this, note that we can only have an accepting SCC in \mathcal{A}_i if it is also an SCC in the graph \mathcal{G} built by the streamlining construction when starting to consider color i (as all transitions from and to states not in \mathcal{G} at that point of the construction have been assigned colors strictly smaller than i in the streamlined automaton).

But then, either the minimal transition color in the SCC has the same parity as i , and then it is lowered to i in the streamlining construction, or it does not. In the latter case, the streamlining construction lowers the color of such a minimal transition color in the SCC to $i - 1$ by the third step of the construction before actually considering color i , contradicting the assumption that the SCC has only accepting transitions in \mathcal{A}_i (as transitions with color smaller than i are not accepting in \mathcal{A}_i by Def. 4).

Since we now know that the minimal color in the maximal accepting SCC in \mathcal{A}_i is i in the streamlined automaton \mathcal{A} , we can insert into w , in every position in $j \in J$, a suffix language invariant string, whose partial run is a cycle in both \mathcal{A}_i and \mathcal{A} , in the latter case with minimal color i . Therefore, the resulting word w' is accepted by \mathcal{A} if, and only if, i is even. As i and $i - 1$ have a different parity, condition (c2) cannot hold for w' .

However, condition (c1) also cannot hold: The accepting run of \mathcal{A}_i on w' is also an accepting run of \mathcal{A}_{i-1} , so its natural color is at least $i - 1$ by our inductive hypothesis. Taking the falsification of both (c1) and (c2) together, we obtain that w cannot have a natural color of at most $i - 1$. (contradiction)

Substep 2: Finally, we show that a word $w = w_0w_1\dots$ that is rejected by \mathcal{A}_i has natural color of at most $i - 1$. If the word is rejected by \mathcal{A}_{i-1} , then the natural color is at most $i - 2$ by our inductive hypothesis, and there is nothing more to be shown. So we henceforth only need to consider the case that w is accepted by \mathcal{A}_{i-1} but not \mathcal{A}_i . We define a suitable infinite set of indices $J \subseteq \mathbb{N}$.

We first assume for contradiction that there is a position $p > 0$ in the word such that, for all positions $p' > p$, there is a run $\pi = q_0q_1\dots$ of \mathcal{A}_i where $(q_j, w_j, q_{j+1}, 2) \in \delta_i$ is, for all $p \leq j < p'$, an accepting transition in \mathcal{A}_i . If no such position exists, the finitely branching tree of runs of \mathcal{A}_i on w that is pruned at all non-accepting positions after level p is infinite, and therefore has an infinite path. (contradiction to $w \notin \mathcal{L}(\mathcal{A}_i)$)

Using this observation, we fix an infinite ascending chain $0 < p_0 < p_1 < p_2 \dots$, such that, for all $j \geq 0$, no run has only accepting transitions in any segment $q_{p_j}q_{p_j+1}\dots q_{p_{j+1}}$.

We note that inserting suffix language invariant strings in positions of $J = \{p_j \mid j \in \mathbb{N}\}$ does not change that these segments have this property; consequently, any word w' that results from such insertions is still rejected by \mathcal{A}_i . We fix such a word w' .

Let c be the maximal color such that $w' = w'_0w'_1w'_2\dots$ is in the language of \mathcal{A}_c . As $w' \notin \mathcal{L}(\mathcal{A}_i)$, $c < i$. If $c < i - 1$, then its natural color is $c < i - 1$ by induction hypothesis. If it is $c = i - 1$, then the natural color must be at least $i - 1$ by induction hypothesis. Moreover, \mathcal{A}_{i-1} has an accepting run $\pi' = q_0q'_1q'_2\dots$ on w' . Let p be a position in this run such that, for all $j > p$, the transitions $(q'_j, w'_j, q'_{j+1}, 2) \in \delta_{i-1}$ are accepting transitions of \mathcal{A}_{i-1} . Noting that π' is a rejecting run of \mathcal{A}_i , this entails that the lowest color that occurs infinitely often in the run $q'_p q'_{p+1} q'_{p+2} \dots$ of $\mathcal{A}_{q'_p}$ on $w'_p w'_{p+1} w'_{p+2} \dots$ is $i - 1$. Thus $w'_p w'_{p+1} w'_{p+2} \dots$ is accepted by $\mathcal{A}_{q'_p}$, and therefore w' is accepted by \mathcal{A} if, and only if, i is odd.

This concludes the proof that the natural color of w is $i - 1$. ◀

Taking the results above together, we have obtained a construction for a language recognised by a given deterministic parity automaton that provides a sequence of co-Büchi automata that encode which word is at home in which color. We first compute a structured form of this deterministic parity automaton, then streamline it (Def. 2), and finally split the resulting parity automaton into the co-Büchi automata according to Def. 4. All three steps can be implemented to run in time polynomial in the size of the input automaton. Furthermore, since the state spaces of the co-Büchi automata are the same as the one in the parity automaton, they cannot be larger than the original parity automaton.

Since the split into co-Büchi automata is canonical, and the co-Büchi automata themselves can be made canonical (and minimal) using the existing polynomial-time construction from Abu Radi and Kupferman [1], we overall obtain a canonical representation of the language that the deterministic parity automaton we started with represents. Moreover, it can be computed in polynomial time.

4.3 Reading off Natural Colors from Streamlined Automata

The construction so far has the property that it does not immediately provide a direct way of computing the natural color of a word (yet). Given a word, we can check which of the co-Büchi automata built according to Def. 4 accepts the word to compute the natural color of a word (for the given language), but since they are good-for-games rather than deterministic, this is somewhat cumbersome.

The results above, however, allow us to also define a more direct way to determining the natural color of a word (with respect to a given language), as we show below as a side-result.

We define a *co-run* of a deterministic automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0)$ on a word $w = w_0w_1w_2\dots$ with a run $\pi = q_0q_1q_2\dots$ as a sequence $\pi' = q_0q_1\dots q_{p-1}q_pq'_{p+1}q'_{p+2}q'_{p+3}\dots$ for some $p > 0$, such that $\pi'' = q'_p q'_{p+1} q'_{p+2} q'_{p+3} \dots$ is the run of $\mathcal{A}_{q'_p}$ on the word $w' = w_p w_{p+1} w_{p+2} w_{p+3} \dots$ for some state q'_p that is language equivalent to q_p ($q'_p \sim_{\mathcal{A}} q_p$).

The color of the set of co-runs for a word w is defined to be the maximal dominating color c that occurs infinitely often on some co-run of w .

► **Lemma 8.** *Let \mathcal{A} be a streamlined automaton. Then the color of the set of co-runs of a word w is c if w is in the language of \mathcal{A}_c , but not in the language of \mathcal{A}_{c+1} .*

Proof. A co-run of \mathcal{A} on w with dominating color c is an accepting run of \mathcal{A}_c .

An accepting run $\pi' = q'_0 q'_1 \dots q'_{p-1} q'_p q'_{p+1} q'_{p+2} q'_{p+3} \dots$ of \mathcal{A}_{c+1} has some position p from which point onward only accepting transitions (which all have color $> c$ in \mathcal{A}) are taken. \mathcal{A} therefore has a co-run $\pi' = q_0 q_1 \dots q_{p-1} q_p q'_{p+1} q'_{p+2} q'_{p+3} \dots$, whose dominating color is $> c$. (contradiction) ◀

By combining this lemma with the previous theorem, we get the following corollary.

► **Corollary 9.** *Let \mathcal{A} be a streamlined automaton. Then the color of the set of co-runs of a word w is its natural color for $\mathcal{L}(\mathcal{A})$.*

Co-runs are closely related to the GCAs we have defined earlier. The difference is that the “new” transitions to language equivalent states can be used only once along a run. This allows for having a definition on the deterministic automaton (without falling back on good-for-games automata), and is therefore simpler. It also binds the proofs together: the *minimal prefix* from the proof of Theorem 6 corresponds to the shortest prefix at which this single transition to a language equivalent state can be taken. While this provides a more direct connection to the color, the restriction to taking these transitions at most once loses the good-for-games property: as a wrong decision cannot be corrected, access to the remainder of the run may be required. This makes GCAs more attractive, as co-Büchi languages have canonical representatives.

Note that, for an ultimately periodic word (i.e., a word of the form $w = uv^\omega$ for $u, v \in \Sigma^*$), the highest color among the co-runs can be computed in the time polynomial in the number of states, which allows reading off the natural color of a word from a (streamlined) DPA without building the canonical representation of the language of the DPA.

5 Related Work

There already exists an indirect normal form of ω -languages. Every ω -regular language can be represented as a deterministic finite automaton (DFA): This DFA accepts words of the form $u\$v$ for which the *ultimately periodic word* uv^ω is in the ω -language to be represented. Calbrix et al. [3] showed how to compute such a DFA from a given nondeterministic Büchi automaton (to which a deterministic parity automaton is easy to translate). The minimized DFA for this *lasso language* over finite words can then serve as a canonical representation of the ω -language, as two automata representing ω -regular languages encode the same language if, and only if, they accept the same ultimately periodic words.

DFAs that capture ultimately periodic words have furthermore been refined to *families* of DFAs [2] that can be more succinct and share the property to consist of multiple sub-automata with the sequences of co-Büchi automata that we define in this paper.

Such DFAs (or families of DFAs), however, do not implement a core idea of automata, namely to read a word letter-by-letter and to encode the relevant information about the letters of the word already read in a state. It is also unknown how the sizes of such automata relate to the size of a minimal deterministic parity automaton representing the language. Finally, neither DFAs for lasso languages nor families of DFAs have a direct connection to the complexity of a language, as it is, for example, captured by the minimal number of colors used in deterministic parity automata.

6 Conclusion

A classical question in the theory of automata is how to define *canonical automata*.

In this paper, we have taken a step back, and looked at the question of how to define canonical automata for ω -regular languages from a new angle. For a language to be defined canonically by, say, a canonical deterministic parity automaton, a word first and foremost needs a natural color. What should this color be?

Picking the flowers of Niwiński and Walukiewicz [10] as a starting point, we have lifted the same principle to languages in a way that is oblivious to the automaton used.

For an ω -regular language L , we look at a sequence of languages $L_0 \supset L_1 \supset L_2 \dots \supset L_c$ such that

- L_0 is the universal language;
- L_1 is the smallest co-Büchi language contained in L_0 and containing $\overline{L} \cap L_0$ such that $\overline{L_1}$ is closed under insertion;
- L_2 is the smallest co-Büchi language contained in L_1 and containing $L \cap L_1$ such that $\overline{L_2}$ is closed under insertion;
- L_3 is the smallest co-Büchi language contained in L_2 and containing $\overline{L} \cap L_2$ such that $\overline{L_3}$ is closed under insertion; etc.

The *closure under insertion* of a set $\overline{L_i}$ refers to the existence of an infinite set of positions (for each word) at which arbitrary suffix language invariant finite letter sequences can be added to the word without the resulting word leaving $\overline{L_i}$.

This assigns a *color* to each infinite word w , both in the language and outside of it, purely defined by the maximal i such that $w \in L_i$ – we say that w has a natural color of i (for L). As one would expect for a parity condition, $w \in L$ if, and only if, the natural color i of w is even.

The natural color of w for L is thus defined without reference to an automaton (or any other representation of the language). Yet, L is recognized by a deterministic parity automaton with maximal color c if, and only if, L_{c+1} is empty. This sets the minimal maximal color in the automaton to the maximal i such that L_i is non-empty, which further connects this construction to deterministic automata.

We infer these languages by turning a single *streamlined* deterministic parity automaton, which is cheap and easy to obtain from any deterministic parity automaton that recognizes L : L_i contains the set of words whose co-runs have colors of at least i .

Beyond providing evidence *that* a word is in the language, it also provides insight into *why* it is part of this language by peeling off co-Büchi languages of accepted and rejected words layer by layer.

Returning to our chain of languages, this answers the “why co-Büchi?” question that begs to be asked. Each L_c is a co-Büchi language, which is an ideal basis for a natural representation, because co-Büchi languages have recently obtained a canonical representation, albeit not for deterministic automata, but for *good-for-games co-Büchi automata* with transition-based acceptance (GCAs). We can use this to obtain a natural representation for the languages that allow us to identify the natural color of a word.

References



- 1 Bader Abu Radi and Orna Kupferman. Minimizing GFG transition-based automata. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 100:1–100:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.100.
- 2 Dana Angluin and Dana Fisman. Learning regular omega languages. In Peter Auer, Alexander Clark, Thomas Zeugmann, and Sandra Zilles, editors, *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8776 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2014. doi:10.1007/978-3-319-11662-4_10.
- 3 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational w -languages. In *9th International Conference on Mathematical Foundations of Programming Semantics (MFPS)*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993. doi:10.1007/3-540-58027-1_27.
- 4 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO Theor. Informatics Appl.*, 33(6):495–506, 1999. doi:10.1051/ita:1999129.
- 5 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006. doi:10.1007/11874683_26.
- 6 John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971. doi:10.1016/B978-0-12-417750-5.50022-1.
- 7 Denis Kuperberg and Michal Skrzypczak. On determinisation of good-for-games automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6_24.
- 8 Christof Löding. Efficient minimization of deterministic weak omega-automata. *Inf. Process. Lett.*, 79(3):105–109, 2001. doi:10.1016/S0020-0190(00)00183-6.

- 9 Christof Löding and Andreas Tollkötter. State space reduction for parity automata. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPICs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CSL.2020.27.
- 10 Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 320–331. Springer, 1998. doi:10.1007/BFb0028571.
- 11 Sven Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPICs*, pages 400–411. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICs.FSTTCS.2010.400.
- 12 Sven Schewe. Minimising good-for-games automata is NP-complete. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPICs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.FSTTCS.2020.56.

Synthesizing Dominant Strategies for Liveness

Bernd Finkbeiner  

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Noemi Passing  

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

Reactive synthesis automatically derives a strategy that satisfies a given specification. However, requiring a strategy to meet the specification in every situation is, in many cases, too hard of a requirement. Particularly in compositional synthesis of distributed systems, individual winning strategies for the processes often do not exist. Remorsefree dominance, a weaker notion than winning, accounts for such situations: dominant strategies are only required to be as good as any alternative strategy, i.e., they are allowed to violate the specification if no other strategy would have satisfied it in the same situation. The composition of dominant strategies is only guaranteed to be dominant for safety properties, though; preventing the use of dominance in compositional synthesis for liveness specifications. Yet, safety properties are often not expressive enough. In this paper, we thus introduce a new winning condition for strategies, called delay-dominance, that overcomes this weakness of remorsefree dominance: we show that it is compositional for many safety and liveness specifications, enabling a compositional synthesis algorithm based on delay-dominance for general specifications. Furthermore, we introduce an automaton construction for recognizing delay-dominant strategies and prove its soundness and completeness. The resulting automaton is of single-exponential size in the squared length of the specification and can immediately be used for safety synthesis procedures. Thus, synthesis of delay-dominant strategies is, as synthesis of winning strategies, in $2EXPTIME$.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Dominant Strategies, Compositional Synthesis, Reactive Synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.37

Related Version *Full Version*: <https://arxiv.org/abs/2210.01660> [19]

Funding Supported by DFG grant 389792660 as part of TRR 248 (CPEC) and by ERC grant 683300 (OSARES). N. Passing carried out this work as PhD candidate at Saarland University, Germany.

1 Introduction

Reactive synthesis is the task of automatically deriving a strategy that satisfies a formal specification, e.g., given in LTL [31], in *every* situation. Such strategies are called winning. In many cases, however, requiring the strategy to satisfy the specification in every situation is too hard of a requirement. A prominent example is the compositional synthesis of distributed systems consisting of several processes. Compositional approaches for distributed synthesis [27, 13, 14, 15, 18] break down the synthesis task for the whole system into several smaller ones for the individual processes. This is necessary due to the general undecidability [33] of distributed synthesis and the non-elementary complexity [20] for decidable cases: non-compositional distributed synthesis approaches [22, 21] suffer from a severe state space explosion problem and are thus not feasible for larger systems. However, winning strategies rarely exist when considering the processes individually in the smaller subtasks of compositional synthesis since usually the processes need to collaborate in order to achieve the overall system's correctness. For instance, a particular input sequence may prevent the satisfaction of the specification no matter how a single process reacts, yet, the other processes of the system ensure in the interplay of the whole system that this input sequence will never be produced.



© Bernd Finkbeiner and Noemi Passing;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 37; pp. 37:1–37:19

Leibniz International Proceedings in Informatics



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

Remorsefree dominance [9], a weaker notion than winning, accounts for such situations. A dominant strategy is allowed to violate the specification as long as no other strategy would have satisfied it in the same situation. Hence, a dominant strategy is a best-effort strategy as we do not blame it for violating the specification if the violation is not its fault. Searching for dominant strategies rather than winning ones allows us to find strategies that do not necessarily satisfy the specification in all situations but in all that are *realistic* in the sense that they occur in the interplay of the processes if all of them play best-effort strategies.

The parallel composition of dominant strategies, however, is only guaranteed to be dominant for safety properties [10]. For liveness specifications, in contrast, dominance is not a compositional notion and thus not suitable for compositional synthesis. Consider, for example, a system with two processes p_1 and p_2 sending messages to each other, denoted by atomic propositions m_1 and m_2 , respectively. Both processes are required to send their message eventually, i.e., $\varphi = \diamond m_1 \wedge \diamond m_2$. For p_i , it is dominant to wait for the other process to send the message m_{3-i} before sending its own message m_i : if p_{3-i} sends its message eventually, p_i does so as well, satisfying φ . If p_{3-i} never sends its message, φ is violated, no matter how p_i reacts, and thus the violation of φ is not p_i 's fault. Combining these strategies for p_1 and p_2 , however, yields a system that never sends any message since both processes wait indefinitely for each other, while there clearly exist strategies for the whole system that satisfy φ .

Bounded dominance [10] is a variant of remorsefree dominance that ensures compositionality of general properties. Intuitively, it reduces every specification φ to a safety property by introducing a measure of the strategy's progress with respect to φ , and by bounding the number of non-progress steps, i.e., steps in which no progress is made. Yet, bounded dominance has two major disadvantages: (i) it requires a concrete bound on the number of non-progress steps, and (ii) not every bounded dominant strategy is dominant: if the bound n is chosen too small, every strategy, also a non-dominant one, is trivially n -dominant.

In this paper, we introduce a new winning condition for strategies, called *delay-dominance*, that builds upon the ideas of bounded dominance but circumvents the aforementioned weaknesses. Similar to bounded dominance, it introduces a progress measure on strategies. However, it does not require a concrete bound on the number of non-progress steps but relates such steps in the potentially delay-dominant strategy s to non-progress steps in an alternative strategy t : intuitively, s delay-dominates t if, whenever s makes a non-progress step, t makes a non-progress step *eventually* as well. A strategy s is then delay-dominant if it delay-dominates every other strategy t . In this way, we ensure that a delay-dominant strategy satisfies the specification "faster" than all other strategies in all situations in which the specification can be satisfied. Delay-dominance considers specifications given as alternating co-Büchi automata. Non-progress steps with respect to the automaton are those that enforce a visit of a rejecting state in all run trees. We introduce a two-player game, the so-called *delay-dominance game*, which is vaguely leaned on the delayed simulation game for alternating Büchi automata [24], to formally define delay-dominance: the winner of the game determines whether or not a strategy s delay-dominates a strategy t on a given input sequence.

We (i) show that every delay-dominant strategy is also remorsefree dominant, and (ii) introduce a criterion for automata such that, if the criterion is satisfied, compositionality of delay-dominance is guaranteed. The criterion is satisfied for many automata; both ones describing safety properties and ones describing liveness properties. Thus, delay-dominance overcomes the weaknesses of both remorsefree and bounded dominance. Note that since delay-dominance relies, as bounded dominance, on the automaton structure, there are realizable specifications for which no delay-dominant strategy exists. Yet, we experienced that this rarely occurs in practice when constructing the automaton from an LTL formula with standard algorithms. Moreover, if a delay-dominant strategy exists, it is guaranteed to be

winning if the specification is realizable. Hence, the parallel composition of delay-dominant strategies for all processes in a distributed system is winning for the whole system as long as the specification is realizable and as long as the compositionality criterion is satisfied. Therefore, delay-dominance is a suitable notion for compositional synthesis.

We thus introduce a synthesis approach for delay-dominant strategies that immediately enables a compositional synthesis algorithm for distributed systems, namely synthesizing delay-dominant strategies for the processes separately. We present the construction of a universal co-Büchi automaton $\mathcal{A}_{\mathcal{A}\varphi}^{dd}$ from an LTL formula φ that recognizes delay-dominant strategies. $\mathcal{A}_{\mathcal{A}\varphi}^{dd}$ can immediately be used for safrless synthesis [28] approaches such as bounded synthesis [22] to synthesize delay-dominant strategies. We show that the size of $\mathcal{A}_{\mathcal{A}\varphi}^{dd}$ is single-exponential in the squared length of φ . Thus, synthesis of delay-dominant strategies is, similar to synthesis of winning or remorsefree dominant strategies, in 2EXPTIME.

Related Work. *Remorsefree dominance* has first been introduced for reactive synthesis in [9]. Dominant strategies have been utilized for compositional synthesis of safety properties [10]. Building up on this work, a compositional synthesis algorithm, that finds solutions in more cases by incrementally synthesizing individual dominant strategies, has been developed [16]. Both algorithms suffer from the non-compositionality of dominant strategies for liveness properties. *Bounded dominance* [10], a variant of dominance that introduces a bound on the number of steps in which a strategy does not make progress with respect to the specification, solves this problem. However, it requires a concrete bound on the number of non-progress steps. Moreover, a bounded dominant strategy is not necessarily dominant.

Good-enough synthesis [1, 29] follows a similar idea as dominance. It is thus not compositional for liveness properties either. In good-enough synthesis, conjuncts of the specification can be marked as *strong*. If the specification is unrealizable, a good-enough strategy needs to satisfy the strong conjuncts while it may violate the other ones. Thus, dominance can be seen as the special case of good-enough synthesis in which no conjuncts are marked as strong. Good-enough synthesis can be extended to a multi-valued correctness notion [1].

Synthesis under environment assumptions is a well-studied problem that also aims at relaxing the requirements on a strategy. There, explicit assumptions on the environment are added to the specification. These assumptions can be LTL formulas restricting the possible input sequences (see, e.g., [7, 5]) or environment strategies (see, e.g., [2, 3, 17, 18]). The assumptions can also be conceptual such as assuming that the environment is rational (see, e.g., [23, 26, 6, 8]). Synthesis under environment assumptions is orthogonal to the synthesis of dominant strategies and good-enough synthesis since it requires an explicit assumption on the environment, while the latter two approaches rely on implicit assumptions.

2 Preliminaries

Notation. Given an infinite word $\sigma = \sigma_0\sigma_1 \dots \in (2^\Sigma)^\omega$, we denote the prefix of length $t + 1$ of σ with $\sigma|_t := \sigma_0 \dots \sigma_t$. For σ and a set $X \subseteq \Sigma$, let $\sigma \cap X := (\sigma_0 \cap X)(\sigma_1 \cap X) \dots \in (2^X)^\omega$. For $\sigma \in (2^\Sigma)^\omega$, $\sigma' \in (2^{\Sigma'})^\omega$ with $\Sigma \cap \Sigma' = \emptyset$, we define $\sigma \cup \sigma' := (\sigma_0 \cup \sigma'_0)(\sigma_1 \cup \sigma'_1) \dots \in (2^{\Sigma \cup \Sigma'})^\omega$. For a k -tuple a , we denote the j -th component of a with $j(a)$. We represent a Boolean formula $\bigvee_i \bigwedge_j c_{i,j}$ in disjunctive normal form (DNF) also in its set notation $\bigcup_i \{ \bigcup_j \{ c_{i,j} \} \}$.

LTL. Linear-time temporal logic (LTL) [31] is a standard specification language for linear-time properties. Let Σ be a finite set of atomic propositions and let $a \in \Sigma$. The syntax of LTL is given by $\varphi, \psi ::= a \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \psi$. We define *true* = $a \vee \neg a$, *false* = $\neg \text{true}$, $\diamond\varphi = \text{true} \mathcal{U} \varphi$, and $\square\varphi = \neg \diamond \neg\varphi$ as usual. We use the standard semantics. The language $\mathcal{L}(\varphi)$ of an LTL formula φ is the set of infinite words that satisfy φ .

Non-Alternating ω -Automata. Given a finite alphabet Σ , a Büchi (resp. co-Büchi) automaton $\mathcal{A} = (Q, Q_0, \delta, F)$ over Σ consists of a finite set of states Q , an initial state $q_0 \in Q$, a transition relation $\delta : Q \times 2^\Sigma \times Q$, and a set of accepting (resp. rejecting) states $F \subseteq Q$. For an infinite word $\sigma = \sigma_0\sigma_1 \dots \in (2^\Sigma)^\omega$, a run of \mathcal{A} induced by σ is an infinite sequence $q_0q_1 \dots \in Q^\omega$ of states with $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i \geq 0$. A run is accepting if it contains infinitely many accepting states (resp. only finitely many rejecting states). A nondeterministic (resp. universal) automaton \mathcal{A} accepts a word σ if some run is accepting (resp. all runs are accepting). The language $\mathcal{L}(\mathcal{A})$ of \mathcal{A} is the set of all accepted words. We consider nondeterministic Büchi automata (NBAs) and universal co-Büchi automata (UCAs).

Alternating ω -Automata. An alternating Büchi (resp. co-Büchi) automaton (ABA resp. ACA) $\mathcal{A} = (Q, q_0, \delta, F)$ over a finite alphabet Σ consists of a finite set of states Q , an initial state $q_0 \in Q$, a transition function $\delta : Q \times 2^\Sigma \rightarrow \mathbb{B}^+(Q)$, where $\mathbb{B}^+(Q)$ is the set of positive Boolean formulas over Q , and a set of accepting (resp. rejecting) states $F \subseteq Q$. We assume that the elements of $\mathbb{B}^+(Q)$ are given in DNF. Runs of \mathcal{A} are Q -labeled trees: a tree \mathcal{T} is a prefix-closed subset of \mathbb{N}^* . The children of a node $x \in \mathcal{T}$ are $c(x) = \{x \cdot d \in \mathcal{T} \mid d \in \mathbb{N}\}$. An X -labeled tree (\mathcal{T}, ℓ) consists of a tree \mathcal{T} and a labeling function $\ell : \mathcal{T} \rightarrow X$. A branch of (\mathcal{T}, ℓ) is a maximal sequence $\ell(x_0)\ell(x_1) \dots$ with $x_0 = \varepsilon$ and $x_{i+1} \in c(x_i)$ for $i \geq 0$. A run tree of \mathcal{A} induced by $\sigma \in (2^\Sigma)^\omega$ is a Q -labeled tree (\mathcal{T}, ℓ) with $\ell(\varepsilon) = q_0$ and, for all $x \in \mathcal{T}$, $\{\ell(x') \mid x' \in c(x)\} \in \delta(\ell(x), \sigma_{|x|})$. A run tree is accepting if every infinite branch contains infinitely many accepting states (resp. only finitely many rejecting states). \mathcal{A} accepts σ if there is some accepting run tree. The language $\mathcal{L}(\mathcal{A})$ of \mathcal{A} is the set of all accepted words.

Two-Player Games. An arena is a tuple $\mathbb{A} = (V, V_0, V_1, v_0, E)$, where V, V_0, V_1 are finite sets of positions with $V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$, $v_0 \in V$ is the initial position, $E \subseteq V \times V$ is a set of edges such that $\forall v \in V. \exists v' \in V. (v, v') \in E$. Player i controls positions in V_i . A game $\mathcal{G} = (\mathbb{A}, W)$ consists of an arena \mathbb{A} and a winning condition $W \subseteq V^\omega$. A play is an infinite sequence $\rho \in V^\omega$ such that $(\rho_i, \rho_{i+1}) \in E$ for all $i \in \mathbb{N}$. The player owning a position chooses the edge on which the play is continued. A play ρ is initial if $\rho_0 = v_0$ holds. It is winning for Player 0 if $\rho \in W$ and winning for Player 1 otherwise. A strategy for Player i is a function $\tau : V^*V_i \rightarrow V$ such that $(v, v') \in E$ whenever $\tau(w, v) = v'$ for some $w \in V^*, v \in V_i$. A play ρ is consistent with a strategy τ if, for all $j \in \mathbb{N}$, $\rho_j \in V_i$ implies $\rho_{j+1} = \tau(\rho|_j)$. A strategy for Player i is winning if all initial and consistent plays are winning for Player i .

System Architectures. An architecture is a tuple $A = (P, \Sigma, \text{inp}, \text{out})$, where P is a set of processes consisting of the environment env and a set $P^- = P \setminus \{\text{env}\}$ of n system processes, Σ is a set of Boolean variables, $\text{inp} = \langle I_1, \dots, I_n \rangle$ assigns a set $I_j \subseteq \Sigma$ of input variables to each $p_j \in P^-$, and $\text{out} = \langle O_{\text{env}}, O_1, \dots, O_n \rangle$ assigns a set $O_j \subseteq \Sigma$ of output variables to each $p_j \in P$. For all $p_j, p_k \in P^-$ with $j \neq k$, $I_j \cap O_k = \emptyset$ and $O_j \cap O_k = \emptyset$ hold. The variables Σ_j of $p_j \in P^-$ are given by $\Sigma_j = I_j \cup O_j$. The inputs I , outputs O , and variables Σ of the whole system are defined by $X = \bigcup_{p_j \in P^-} X_j$ for $X \in \{I, O, \Sigma\}$. A is called distributed if $|P^-| \geq 2$. In the remainder of this paper, we assume that a distributed architecture is given.

Process Strategies. A strategy for process p_i is a function $s_i : (2^{I_i})^* \rightarrow 2^{O_i}$ mapping a history of inputs to outputs. We model s_i as a Moore machine $\mathcal{M}_i = (T, t_0, \tau, o)$ consisting of a finite set of states T , an initial state $t_0 \in T$, a transition function $\tau : T \times 2^{I_i} \rightarrow T$, and a labeling function $o : T \rightarrow 2^{O_i}$. For a sequence $\gamma = \gamma_0\gamma_1 \dots \in (2^{I_i})^\omega$, \mathcal{M}_i produces a path $(t_0, \gamma_0 \cup o(t_0))(t_1, \gamma_1 \cup o(t_1)) \dots \in (T \times 2^{I_i \cup O_i})^\omega$, where $\tau(t_j, \gamma_j) = t_{j+1}$. The projection

of a path to the variables is called a trace. The trace produced by \mathcal{M}_i on $\gamma \in (2^{I_i})^\omega$ is called the computation of s_i on γ , denoted $\text{comp}(s_i, \gamma)$. We say that s_i is winning for an LTL formula φ , denoted $s_i \models \varphi$, if $\text{comp}(s_i, \gamma) \models \varphi$ holds for all input sequences $\gamma \in (2^{I_i})^\omega$. Overloading notation with two-player games, we call a process strategy simply a strategy whenever the context is clear. The parallel composition $\mathcal{M}_i \parallel \mathcal{M}_j$ of two Moore machines $\mathcal{M}_i = (T_i, t_0^i, \tau_i, o_i)$, $\mathcal{M}_j = (T_j, t_0^j, \tau_j, o_j)$ for $p_i, p_j \in P^-$ is the Moore machine (T, t_0, τ, o) with inputs $(I_i \cup I_j) \setminus (O_i \cup O_j)$ and outputs $O_i \cup O_j$ as well as $T = T_i \times T_j$, $t_0 = (t_0^i, t_0^j)$, $\tau((t, t'), \iota) = (\tau_i(t, (\iota \cup o_j(t')) \cap I_i), \tau_j(t', (\iota \cup o_i(t)) \cap I_j))$, and $o((t, t')) = o_i(t) \cup o_j(t')$.

Synthesis. Given a specification φ , synthesis derives strategies s_1, \dots, s_n for the system processes such that $s_1 \parallel \dots \parallel s_n \models \varphi$, i.e., such that the parallel composition of the strategies satisfies φ for all input sequences generated by the environment. If such strategies exist, φ is called realizable. Bounded synthesis [22] additionally bounds the size of the strategies. The search for strategies is encoded into a constraint system that is satisfiable if, and only if, φ is realizable for the size bound. There are SMT, SAT, QBF, and DQBF encodings [22, 11, 4]. We consider a compositional synthesis approach that synthesizes strategies for the processes separately. Thus, outputs produced by the other system processes are treated similar to the environment outputs, namely as part of the input sequence of the considered process. Nevertheless, compositional synthesis derives strategies such that $s_1 \parallel \dots \parallel s_n \models \varphi$ holds.

3 Dominant Strategies and Liveness Properties

Given a specification φ , the naïve compositional synthesis approach is to synthesize strategies s_1, \dots, s_n for the system processes such that $s_i \models \varphi$ holds for all $p_i \in P^-$. Then, it follows immediately that $s_1 \parallel \dots \parallel s_n \models \varphi$ holds as well. However, since winning strategies are required to satisfy φ for every input sequence, usually no such individual winning strategies exist due to complex interconnections in the system. Therefore, the naïve approach fails in many cases. The notion of *remorsefree dominance* [9], in contrast, has been successfully used in compositional synthesis [10, 16]. The main idea is to synthesize *dominant* strategies for the system processes separately instead of winning ones. Dominant strategies are, in contrast to winning strategies, allowed to violate the specification for some input sequence if no other strategy would have satisfied it in the same situation. Thus, remorsefree dominance is a weaker requirement than winning and therefore individual dominant strategies exist for more systems. Formally, remorsefree dominant strategies are defined as follows:

► **Definition 1** (Dominant Strategy [10]). *Let φ be an LTL formula. Let s and t be strategies for process p_i . Then, t is dominated by s , denoted $t \preceq s$, if for all input sequences $\gamma \in (2^{I_i})^\omega$ either $\text{comp}(s, \gamma) \models \varphi$ or $\text{comp}(t, \gamma) \not\models \varphi$ holds. Strategy s is called dominant for φ if $t \preceq s$ holds for all strategies t for process p_i .*

Intuitively, a strategy s dominates a strategy t if it is *at least as good* as t . It is dominant for φ if it is at least as good as *every other possible strategy* and thus if it is *as good as possible*. As an example, reconsider the message sending system. Let s_i be a strategy for process p_i that outputs m_i in the very first step. It satisfies $\varphi = \diamond m_1 \wedge \diamond m_2$ on all input sequences containing at least one m_{3-i} . On all other input sequences, it violates φ . Let t_i be some alternative strategy. Since no strategy for p_i can influence m_{3-i} , t_i satisfies φ only on input sequences containing at least one m_{3-i} . Yet, s_i satisfies φ for such sequences as well. Hence, s_i dominates t_i and since we chose t_i arbitrarily, s_i is dominant for φ .

Synthesizing dominant strategies rather than winning ones allows us to synthesize strategies for the processes of a distributed system compositionally, although no winning strategies for the individual processes exist. Dominant strategies for the individual processes can then be recomposed to obtain a strategy for the whole system. For safety specifications, the composed strategy is guaranteed to be dominant for the specification as well:

► **Theorem 2** (Compositionality of Dominance for Safety Properties [10]). *Let φ be an LTL formula. Let s_1 and s_2 be dominant strategies for processes p_1 and p_2 , respectively, as well as for φ . If φ is a safety property, then $s_1 \parallel s_2$ is dominant for $p_1 \parallel p_2$ and φ .*

Compositionality is a crucial property for compositional synthesis: it allows for concluding that the parallel composition of the separately synthesized process strategies is indeed a useful strategy for the whole system. Thus, Theorem 2 enables compositional synthesis with dominant strategies for safety properties. For liveness properties, however, the parallel composition of two dominant strategies is not necessarily dominant: consider strategy t_i for p_i in the message sending system that waits for m_{3-i} before sending its own message. This strategy is dominant for φ : for input sequences in which m_{3-i} occurs eventually, t_i sends m_i in the next step, satisfying φ . For all other input sequences, no strategy for p_i can satisfy φ . Yet, the parallel composition of t_1 and t_2 does not send any message; violating φ , while there exist strategies that satisfy φ , e.g., a strategy sending both m_1 and m_2 in the first step.

Bounded dominance [10] is a variant of dominance that is compositional for both safety and liveness properties. Intuitively, it reduces the specification φ to a safety property by introducing a bound on the number of steps in which the strategy *does not make progress* with respect to φ . The progress measure is defined on an equivalent UCA \mathcal{A} for φ . The measure $m_{\mathcal{A}}$ of a process strategy s on an input sequence $\gamma \in (2^{I_i})^\omega$ is then the supremum of the number of rejecting states of the runs of \mathcal{A} induced by $\text{comp}(s, \gamma)$. Thus, a strategy s n -dominates a strategy t for \mathcal{A} and $n \in \mathbb{N}$ if for every $\gamma \in (2^{I_i})^\omega$, either $m_{\mathcal{A}}(\text{comp}(s, \gamma)) \leq n$ or $m_{\mathcal{A}}(\text{comp}(t, \gamma)) > n$ holds. If \mathcal{A} is a safety automaton, then remorsefree dominance and bounded dominance coincide. For liveness specifications, however, they differ.

Yet, bounded dominance does not imply dominance: there are specifications φ with a minimal measure m , i.e., all strategies have a measure of at least m [10]. When choosing a bound $n < m$, every strategy is trivially n -dominant for φ , even non-dominant ones. Hence, the choice of the bound is crucial for bounded dominance. It is not obvious how to determine a *good* bound, though: it needs to be large enough to avoid non-dominant strategies. As the bound has a huge impact on the synthesis time, however, it cannot be chosen too large as otherwise synthesis becomes infeasible. Especially for specifications with several complex dependencies between processes, it is hard to determine a proper bound. Therefore, bounded dominance is not a suitable notion for compositional synthesis for liveness properties. In the remainder of this paper, we introduce a different variant of dominance that implies remorsefree dominance and that ensures compositionality also for many liveness properties.

4 Delay-Dominance

In this section, we introduce a new winning condition for strategies, *delay-dominance*, which resembles remorsefree dominance but ensures compositionality also for many liveness properties. It builds on the idea of bounded dominance to not only consider the satisfaction of the LTL formula φ but to measure progress based on an automaton representation of φ . Similar to bounded dominance, we utilize visits of rejecting states in a co-Büchi automaton. Yet, we use an *alternating* automaton instead of a universal one. Note that delay-dominance

can be equivalently formulated on UCAs, yet, using ACAs allows for more efficient synthesis of delay-dominant strategies (see Section 5). Moreover, we do not require a fixed bound on the number of visits to rejecting states; rather, we relate visits of rejecting states induced by the delay-dominant strategy to visits of rejecting states induced by the alternative strategy.

Intuitively, delay-dominance requires that every visit to a rejecting state in the ACA \mathcal{A} caused by the delay-dominant strategy *is matched* by a visit to a rejecting state caused by the alternative strategy *eventually*. The rejecting states of the ACA \mathcal{A} are closely related to the satisfaction of the LTL specification φ : if infinitely many rejecting states are visited, then φ is not satisfied. Thus, delay-dominance allows a strategy to violate the specification if all alternative strategies violate it as well. Defining delay-dominance on the rejecting states of \mathcal{A} instead of the satisfaction of φ allows for measuring the progress on satisfying the specification. Thus, we can distinguish strategies that wait indefinitely for another process from those that do not: intuitively, a strategy s that waits will visit a rejecting state later than a strategy t that does not. This visit to a rejecting state is then not matched eventually by a visit to a rejecting state in t , preventing delay-dominance of s .

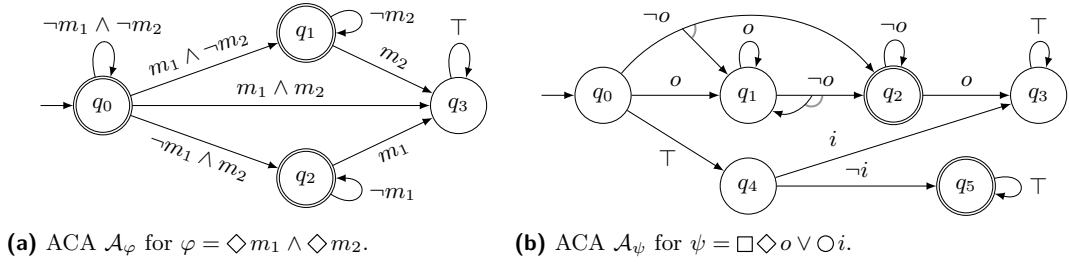
Formally, we present a game-based definition for delay-dominance: we introduce a two-player game, the so-called *delay-dominance game*, which is inspired by the delayed simulation game for alternating Büchi automata [24]. Given an ACA $\mathcal{A} = (Q, q_0, \delta, F)$, two strategies s and t for some process p_i , and an input sequence $\gamma \in (2^{I_i})^\omega$, the delay-dominance game determines whether s delay-dominates t for \mathcal{A} on input γ . Intuitively, the game proceeds in rounds. At the beginning of each round, a pair (p, q) of states $p, q \in Q$ and the number of the iteration $j \in \mathbb{N}$ is given, where p represents a state that is visited by a run of \mathcal{A} induced by $\text{comp}(t, \gamma)$, while q represents a state that is visited by a run of \mathcal{A} induced by $\text{comp}(s, \gamma)$. We call p the *alternative state* and q the *dominant state*. Let $\sigma^s := \text{comp}(s, \gamma)$ and $\sigma^t := \text{comp}(t, \gamma)$. The players Duplicator and Spoiler, where Duplicator takes on the role of Player 0, play as follows: **1.** Spoiler chooses a set $c \in \delta(p, \sigma_j^t)$. **2.** Duplicator chooses a set $c' \in \delta(q, \sigma_j^s)$. **3.** Spoiler chooses a state $q' \in c'$. **4.** Duplicator chooses a state $p' \in c$. The starting pair of the next round is then $((p', q'), j + 1)$. Starting from $((q_0, q_0), 0)$, the players construct an infinite play which determines the winner. Duplicator wins for a play if every rejecting dominant state is matched by a rejecting alternative state eventually.

Both the delay-dominant strategy s and the alternative strategy t may control the non-deterministic transitions of \mathcal{A} , while the universal ones are uncontrollable. Since, intuitively, strategy t is controlled by an opponent when proving that s delay-dominates t , we thus have a change in control for t : for s , Duplicator controls the existential transitions of \mathcal{A} and Spoiler controls the universal ones. For t , Duplicator controls the universal transitions and Spoiler controls the existential ones. Note that the order in which Spoiler and Duplicator make their moves is crucial to ensure that Duplicator wins the game when considering the very same process strategies. By letting Spoiler move first, Duplicator is able to mimic – or *duplicate* – Spoiler’s moves. Formally, the delay-dominance game is defined as follows:

► **Definition 3** (Delay-Dominance game). *Let $\mathcal{A} = (Q, q_0, \delta, F)$ be an ACA. Based on \mathcal{A} , we define the sets $S_\exists = (Q \times Q) \times \mathbb{N}$, $D_\exists = (Q \times Q \times 2^Q) \times \mathbb{N}$, $S_\forall = (Q \times Q \times 2^Q \times 2^Q) \times \mathbb{N}$, and $D_\forall = (Q \times Q \times Q \times 2^Q) \times \mathbb{N}$. Let $\sigma, \sigma' \in (2^{\Sigma_i})^\omega$ be infinite sequences. Then, the delay-dominance game $(\mathcal{A}, \sigma, \sigma')$ is the game $\mathcal{G} = (\mathbb{A}, W)$ defined by $\mathbb{A} = (V, V_0, V_1, v_0, E)$ with $V = S_\exists \cup D_\exists \cup S_\forall \cup D_\forall$, $V_0 = D_\exists \cup D_\forall$, and $V_1 = S_\exists \cup S_\forall$ as well as*

$$E = \{((p, q), j), ((p, q, c), j) \mid c \in \delta(p, \sigma_j)\} \cup \{((p, q, c), j), ((p, q, c, c'), j) \mid c' \in \delta(q, \sigma_j')\} \\ \cup \{((p, q, c, c'), j), ((p, q, c, q'), j) \mid q' \in c'\} \cup \{((p, q, c, q'), j), ((p', q'), j + 1) \mid p' \in c\},$$

and the winning condition $W = \{\rho \in V^\omega \mid \forall j \in \mathbb{N}. f_{\text{dom}}(\rho_j) \in F \rightarrow \exists j' \geq j. f_{\text{alt}}(\rho_{j'}) \in F\}$, where $f_{\text{alt}}(v) := 1(1(v))$ and $f_{\text{dom}}(v) := 2(1(v))$, i.e., $f_{\text{alt}}(v)$ and $f_{\text{dom}}(v)$ map a position v to the alternative state and the dominant state of v , respectively.



■ **Figure 1** Alternating co-Büchi automata \mathcal{A}_φ and \mathcal{A}_ψ . Universal choices are depicted by connecting the transitions with a gray arc. Rejecting states are marked with double circles.

We now define the notion of delay-dominance based on the delay-dominance game. Intuitively, the winner of the game for the computations of two strategies s and t determines whether or not s delay-dominates t on a given input sequence. Similar to remorsefree dominance, we then lift this definition to delay-dominant strategies. Formally:

► **Definition 4** (Delay-Dominant Strategy). *Let \mathcal{A} be an ACA. Let s and t be strategies for process p_i . Then, s delay-dominates t on input sequence $\gamma \in (2^{I_i})^\omega$ for \mathcal{A} , denoted $t \preceq_\gamma^{\mathcal{A}} s$, if Duplicator wins the delay-dominance game $(\mathcal{A}, \text{comp}(t, \gamma), \text{comp}(s, \gamma))$. Strategy s delay-dominates t for \mathcal{A} , denoted $t \preceq^{\mathcal{A}} s$, if $t \preceq_\gamma^{\mathcal{A}} s$ holds for all input sequences $\gamma \in (2^{I_i})^\omega$. Strategy s is delay-dominant for \mathcal{A} if, for every alternative strategy t for p_i , $t \preceq^{\mathcal{A}} s$ holds.*

As an example for delay-dominance, consider the message sending system again. Let s_i be a strategy for process p_i that outputs m_i in the very first step and let t_i be a strategy that waits for m_{3-i} before sending its own message. An ACA \mathcal{A}_φ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ is depicted in Figure 1a. Note that \mathcal{A}_φ is deterministic and thus every sequence induces a single run tree with a single branch. Hence, for every input sequence $\gamma \in (2^{I_i})^\omega$, the moves of both Spoiler and Duplicator are uniquely defined by the computations of t_1 and s_1 on γ , respectively. Therefore, we only provide the state pairs (p, q) of the delay-dominance game, not the intermediate tuples. First, consider an input sequence $\gamma \in (2^{I_1})^\omega$ that contains the very first m_2 at point in time ℓ . Then, the run of \mathcal{A}_φ on $\text{comp}(s_1, \gamma)$ starts in q_0 , moves to q_1 immediately if $\ell > 0$, stays there up to the occurrence of m_2 and then moves to q_3 , where it stays forever. If $\ell = 0$, then the run moves immediately from q_0 to q_3 . The run of $\text{comp}(t_1, \gamma)$, in contrast, stays in q_0 until m_2 occurs, then moves to q_2 and then immediately to q_3 , where it stays forever. Thus, we obtain the unique sequence $(q_0, q_0)(q_0, q_1)^{\ell-1}(q_2, q_3)(q_3, q_3)^\omega$ of state pairs in the delay-dominance game $(\mathcal{A}_\varphi, \text{comp}(t_1, \gamma), \text{comp}(s_1, \gamma))$. The last rejecting alternative state, i.e., a rejecting state induced by $\text{comp}(t_1, \gamma)$ occurs at point in time $\ell + 1$, namely q_2 , while the last rejecting dominant state i.e., a rejecting state induced by $\text{comp}(s_1, \gamma)$, occurs at point in time ℓ , namely q_1 . Thus, $t_1 \preceq_\gamma^{\mathcal{A}_\varphi} s_1$ holds. In fact, $t'_1 \preceq_\gamma^{\mathcal{A}_\varphi} s_1$ holds for all alternative strategies t'_1 for such an input sequence γ since every strategy t'_1 for p_1 induces at least ℓ visits to rejecting states due to the structure of γ . Second, consider an input sequence $\gamma' \in (2^{I_i})^\omega$ that does not contain any m_2 . Then, the run of \mathcal{A}_φ on a computation of any strategy t'_1 on γ' never reaches q_3 and thus only visits rejecting states. Hence, in particular, every visit to a rejecting state induced by $\text{comp}(s_1, \gamma')$ is matched by a visit to a rejecting state induced by $\text{comp}(t'_1, \gamma')$ for all strategies t'_1 . Thus, $t'_1 \preceq_{\gamma'}^{\mathcal{A}_\varphi} s_1$ holds for all alternative strategies t'_1 as well. We can thus conclude that s_1 is delay-dominant for \mathcal{A}_φ , meeting our intuition that s_1 should be allowed to violate φ on input sequences that do not contain any m_2 . Strategy t_1 , in contrast, is remorsefree dominant for φ but not delay-dominant for \mathcal{A}_φ : consider again an input sequence $\gamma \in (2^{I_1})^\omega$ that contains the very first m_2 at point in

time ℓ . For the delay-dominance game $(\mathcal{A}_\varphi, \text{comp}(s_1, \gamma), \text{comp}(t_1, \gamma))$, we obtain the following sequence of state pairs: $(q_0, q_0)(q_1, q_0)^{\ell-1}(q_3, q_2)(q_3, q_3)^\omega$. It contains a rejecting dominant state, i.e., a rejecting state induced by $\text{comp}(t_1, \gamma)$, at point in time $\ell + 1$, while the last rejecting alternative state occurs at point in time ℓ . Hence, t_1 does not delay-dominate s_1 , preventing that it is delay-dominant to wait for the other process indefinitely.

Next, consider the LTL formula $\psi = \square \diamond o \vee \bigcirc i$, where i is an input variable and o is an output variable. An ACA \mathcal{A}_ψ with $\mathcal{L}(\mathcal{A}_\psi) = \mathcal{L}(\psi)$ is depicted in Figure 1b. Note that it has both existential and universal transitions. Consider a process strategy s that outputs o in every step. Let t be some alternative strategy and let γ be some input sequence. Then, Duplicator encounters an existential choice in state q_0 for s in the very first round of the delay-dominance game $(\mathcal{A}_\psi, \text{comp}(t, \gamma), \text{comp}(s, \gamma))$: it can choose to move to q_1 or to q_4 . If Duplicator chooses to move to q_1 , then the only possible successor state in every run of \mathcal{A}_ψ induced by $\text{comp}(s, \gamma)$ is q_1 . Thus, irrespective of Spoiler's moves, the sequence of dominant states in all consistent initial plays is given by $q_0 q_1^\omega$. Since neither q_0 nor q_1 is rejecting, Duplicator wins the game. Therefore, there exists a winning strategy for Duplicator for the game $(\mathcal{A}_\psi, \text{comp}(t, \gamma), \text{comp}(s, \gamma))$ for all t and γ , namely choosing to move to q_1 from q_0 , and thus s is delay-dominant. Second, consider a strategy t that does not output o in the first step but outputs o in every step afterwards. Let γ be an input sequence that does not contain i at the second point in time. Then, Duplicator encounters an existential choice in state q_0 for t in the very first round of the delay-dominance game $(\mathcal{A}_\psi, \text{comp}(s, \gamma), \text{comp}(t, \gamma))$. Yet, if Duplicator chooses the transition from q_0 to q_4 , then every consistent play will contain infinitely many rejecting dominant states since the structure of γ enforces that every consistent play enters q_5 in its dominant state in the next round of the game. Otherwise, i.e., if Duplicator chooses the universal transition to both q_1 and q_2 , then Spoiler decides which of the states is entered. If Spoiler chooses q_2 , then every consistent play visits a rejecting dominant state, namely q_2 , in the second round of the game. If Spoiler further chooses to move from q_0 to q_1 for the alternative strategy s , then, as shown above, no rejecting dominant states are visited in a consistent play at all. Thus, there exists a winning strategy for Spoiler and therefore t is not delay-dominant for \mathcal{A}_ψ .

Recall that one of the main weaknesses of bounded dominance is that every strategy, even a non-dominant one, is trivially n -dominant if the bound n is chosen too small. Every delay-dominant strategy, in contrast, is also remorsefree dominant. The main idea is that a winning strategy τ of Duplicator in the delay-dominance game defines a run tree of the automaton induced by the delay-dominant strategy s such that all branches either visit only finitely many rejecting states or such that all rejecting states are matched eventually with a rejecting state in some branch, which is also defined by τ , of all run trees induced by an alternative strategy. Thus, s either satisfies the specification, or an alternative strategy does not satisfy it either. For the formal proof, we refer the reader to [19].

► **Theorem 5.** *Let φ be an LTL formula. Let \mathcal{A}_φ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. Let s be a strategy for process p_i . If s is delay-dominant for \mathcal{A}_φ , then s is remorsefree dominant for φ .*

Clearly, the converse does not hold. For instance, a strategy in the message sending system that waits for the other process to send its message first is remorsefree dominant for φ but not delay-dominant for the ACA depicted in Figure 1a as pointed out above.

Given an LTL formula φ , for remorsefree dominance it holds that if φ is realizable, then every strategy that is dominant for φ is also winning for φ [10]. This is due to the fact that the winning strategy needs to be taken into account as an alternative strategy for every dominant one, and that remorsefree dominance is solely defined on the satisfaction of the specification. With Theorem 5 the same property follows for delay-dominance.

► **Lemma 6.** *Let φ be an LTL formula. Let \mathcal{A}_φ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. If φ is realizable, then every delay-dominant strategy for \mathcal{A}_φ is winning for φ as well.*

A critical shortcoming of remorsefree dominance is its non-compositionality for liveness properties. This restricts the usage of dominance-based compositional synthesis algorithms to safety specifications, which are in many cases not expressive enough. Delay-dominance, in contrast, is specifically designed to be compositional for more properties: we identified that a crucial requirement for the compositionality of a process property such as remorsefree dominance or delay-dominance is the existence of *bad prefixes* for strategies that do not satisfy the process requirement. Since remorsefree dominance solely considers the satisfaction of the specification φ , a bad prefix for a strategy that is not remorsefree dominant boils down to a bad prefix of $\mathcal{L}(\varphi)$ and therefore compositionality cannot be guaranteed for liveness properties. As delay-dominance takes the ACA representing φ and, in particular, its rejecting states into account, the absence of a bad prefix for $\mathcal{L}(\varphi)$ does not necessarily result in the absence of a bad prefix for delay-dominance. First, we define such bad prefixes formally:

► **Definition 7** (Bad Prefixes for Delay-Dominance). *Let \mathcal{P} be the set of all system processes and all parallel compositions of subsets of system processes. Let I_p and O_p be the sets of inputs and outputs of $p \in \mathcal{P}$. Let \mathcal{A} be an ACA. Then, \mathcal{A} ensures bad prefixes for delay-dominance if, for all $p \in \mathcal{P}$ and all strategies s for p for which there exists some $\gamma \in (2^{I_p})^\omega$ such that $s \not\triangleleft_\gamma^{\mathcal{A}} t$ holds for some alternative strategy t , there is a finite prefix $\eta \in (2^{I_p \cup O_p})^*$ of $\text{comp}(s, \gamma)$ such that for all infinite extensions $\sigma \in (2^{I_p \cup O_p})^\omega$ of η , there is an infinite sequence $\sigma' \in (2^{I_p \cup O_p})^\omega$ with $\sigma \cap I_p = \sigma' \cap I_p$ such that Duplicator loses the delay-dominance game $(\mathcal{A}, \sigma', \sigma)$.*

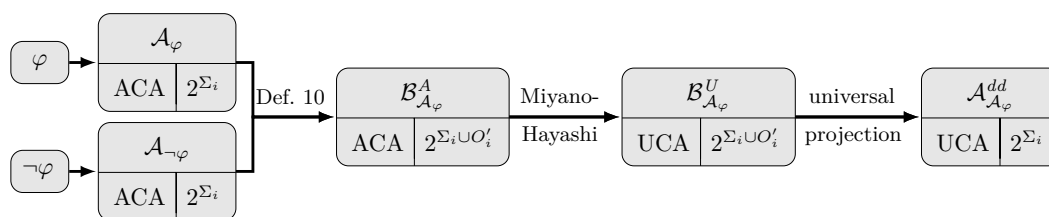
Intuitively, an ACA that ensures bad prefixes for delay-dominance guarantees that for every strategy that is not delay-dominant, there exists a point in time at which its behavior ultimately prevents delay-dominance, irrespective of any future behavior. For more details on bad prefixes for delay-dominance and their existence in co-Büchi automata, we refer to the full version [19]. If an ACA ensures bad prefixes for delay-dominance, compositionality is guaranteed: if the parallel composition of two delay-dominant strategies s_1 and s_2 is not delay-dominant, then the behavior of both processes at the last position of the smallest bad prefix reveals which one of them is responsible for Duplicator losing the game. Note that also both processes can be responsible simultaneously. Since there is an alternative strategy t for the composed system for which Duplicator wins the game, as otherwise $s_1 \parallel s_2$ would be delay-dominant, the strategy of the process p_i which is responsible for Duplicator losing the game cannot be delay-dominant as there is an alternative strategy, namely t restricted to the outputs of p_i , that allows Duplicator to win the game. The formal proof is provided in [19].

► **Theorem 8** (Compositionality of Delay-Dominance). *Let \mathcal{A} be an ACA that ensures bad prefixes for delay-dominance. Let s_1 and s_2 be delay-dominant strategies for \mathcal{A} and processes p_1 and p_2 , respectively. Then, $s_1 \parallel s_2$ is delay-dominant for \mathcal{A} and $p_1 \parallel p_2$.*

From Theorems 5 and 8 it then follows immediately that the parallel composition of two delay-dominant strategies is also remorsefree dominant if the ACA ensures bad prefixes:

► **Corollary 9.** *Let φ be an LTL formula. Let \mathcal{A}_φ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ that ensures bad prefixes for delay-dominance. Let s_1 and s_2 be delay-dominant strategies for \mathcal{A}_φ and processes p_1 and p_2 , respectively. Then, $s_1 \parallel s_2$ is remorsefree dominant for φ and $p_1 \parallel p_2$.*

With Lemma 6 and Theorem 8 we obtain that, given a specification φ and an ACA \mathcal{A}_φ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ that ensures bad prefixes for delay-dominance, the parallel composition of delay-dominant strategies for \mathcal{A}_φ and all processes of a distributed system is winning if φ is realizable. Hence, delay-dominance can be soundly used for dominance-based compositional synthesis approaches when ensuring the bad prefix criterion. In the next section, we thus introduce an automaton construction for synthesizing delay-dominant strategies.



■ **Figure 2** Overview of the construction of a universal co-Büchi automaton $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ recognizing delay-dominant strategies for the alternating co-Büchi automaton \mathcal{A}_φ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. The lower parts of the boxes list the automaton type (alternating or universal) and the alphabet.

5 Synthesizing Delay-Dominant Strategies

In this section, we introduce how delay-dominant strategies can be synthesized using existing tools for synthesizing winning strategies. We focus on utilizing *bounded synthesis* tools such as BoSy [12]. Mostly, we use bounded synthesis as a black box procedure throughout this section. Therefore, we do not go into detail here and refer the interested reader to [22, 11]. A crucial observation regarding bounded synthesis that we utilize, however, is that it translates the given specification φ into an equivalent universal co-Büchi automaton \mathcal{A}_φ and then derives a strategy such that, for every input sequence, the runs of \mathcal{A}_φ induced by the computation of the strategy on the input sequence visit only finitely many rejecting states.

To synthesize delay-dominant strategies instead of winning ones, we can thus use existing bounded synthesis algorithms by replacing the universal co-Büchi automaton \mathcal{A}_φ with one encoding delay-dominance, i.e., with an automaton $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ such that its runs induced by the computations of a delay-dominant strategy on all input sequences visit only finitely many rejecting states. This idea is similar to the approach for synthesizing remorsefree dominant strategies [10, 16]. The automaton for recognizing delay-dominant strategies, however, differs inherently from the one for recognizing remorsefree dominant strategies.

The automaton construction consists of several steps. An overview is given in Figure 2. Since delay-dominance is not defined on the LTL specification φ itself but on an equivalent alternating co-Büchi automaton, we first translate φ into an alternating co-Büchi automaton \mathcal{A}_φ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. For this, we utilize well-known algorithms for translating LTL formulas into equivalent alternating Büchi automata as well as the duality of the Büchi and co-Büchi acceptance condition and of nondeterministic and universal branching. More details on the translation of LTL formulas into alternating co-Büchi automata are provided in [19]. Similarly, we construct an alternating co-Büchi automaton $\mathcal{A}_{\neg\varphi}$ with $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$ from $\neg\varphi$. The centerpiece of the construction is an alternating co-Büchi automaton $\mathcal{B}_{\mathcal{A}_\varphi}^A$ constructed from \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$ that recognizes whether $t \preceq_\gamma^{\mathcal{A}_\varphi} s$ holds for \mathcal{A}_φ , input sequence $\gamma \in (2^{I_i})^\omega$ and strategies s and t for process p_i . The alternating automaton $\mathcal{B}_{\mathcal{A}_\varphi}^A$ is then translated into an equivalent universal co-Büchi automaton $\mathcal{B}_{\mathcal{A}_\varphi}^U$, for example with the Miyano-Hayashi algorithm [30]. Lastly, we translate $\mathcal{B}_{\mathcal{A}_\varphi}^U$ into a universal co-Büchi automaton that accounts for requiring a strategy s to delay-dominate *all* other strategies t and not only a particular one utilizing universal projection. In the remainder of this section, we describe all steps of the construction in detail and prove their correctness.

5.1 Construction of the ACA $\mathcal{B}_{\mathcal{A}_\varphi}^A$

From the two ACAs \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$, we construct an alternating co-Büchi automaton $\mathcal{B}_{\mathcal{A}_\varphi}^A$ that recognizes whether $t \preceq_\gamma^{\mathcal{A}_\varphi} s$ holds for \mathcal{A}_φ , input sequence $\gamma \in (2^{I_i})^\omega$ and process strategies s and t for process p_i . The construction relies on the observation that $t \preceq_\gamma^{\mathcal{A}_\varphi} s$ holds if, and

37:12 Synthesizing Dominant Strategies for Liveness

only if, either (i) $\text{comp}(t, \gamma) \not\models \varphi$ holds or (ii) we have $t \preceq_{\gamma}^{\mathcal{A}_{\varphi}} s$ and every initial play of the delay-dominance game that is consistent with the winning strategy of Duplicator visits only finitely many rejecting dominant states. The proof of this observation is provided in [19]. Therefore, the automaton $\mathcal{B}_{\mathcal{A}_{\varphi}}^A$ consists of two parts, one accounting for (i) and one accounting for (ii), and guesses nondeterministically in the initial state which part is entered. The ACA $\mathcal{A}_{\neg\varphi}$ with $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$ accounts for (i). For (ii), we intuitively build the product of two copies of the ACA \mathcal{A}_{φ} with $\mathcal{L}(\mathcal{A}_{\varphi}) = \mathcal{L}(\varphi)$, one for each of the considered process strategies s and t . Note that similar to the change of control for t in the delay-dominance game, we consider the *dual* transition function of \mathcal{A}_{φ} , i.e., the one where conjunctions and disjunctions are swapped, for the copy of \mathcal{A}_{φ} for t . We keep track of whether we encountered a situation in which a rejecting state was visited for s while it was not for t . This allows for defining the set of rejecting states.

Note that we need to allow for differentiating valuations of output variables computed by s and t on the same input sequence. Therefore, we extend the alphabet of $\mathcal{B}_{\mathcal{A}_{\varphi}}^A$: in addition to the set Σ_i of variables of process p_i , which contains input variables I_i and output variables O_i , we consider the set $O'_i := \{o' \mid o \in O_i\}$ of *primed output variables* of p_i , where every output variable is marked with a prime to obtain a fresh symbol. The set Σ'_i of *primed variables* of p_i is then given by $\Sigma'_i := I_i \cup O'_i$. Intuitively, the output variables O_i depict the behavior of the delay-dominant strategy s , while the primed output variables O'_i depict the behavior of the alternative strategy t . The alphabet of $\mathcal{B}_{\mathcal{A}_{\varphi}}^A$ is then given by $2^{\Sigma_i \cup O'_i}$. This is equivalent to $2^{\Sigma_i \cup \Sigma'_i}$ since the input variables are never primed to ensure that we consider the same input sequence for both strategies. In the following, we use the functions $pr : \Sigma_i \rightarrow \Sigma'_i$ and $unpr : \Sigma'_i \rightarrow \Sigma_i$ to switch between primed variables and normal ones: given a valuation $a \in \Sigma_i$ of variables, $pr(a)$ replaces every output variable $o \in O_i$ occurring in a with its primed version o' . For a valuation $a \in \Sigma'_i$, $unpr(a)$ replaces every primed output variable $o' \in O'_i$ occurring in a with its normal unprimed version o . We extend pr and $unpr$ to finite and infinite sequences as usual. The ACA $\mathcal{B}_{\mathcal{A}_{\varphi}}^A$ is then constructed as follows:

► **Definition 10.** Let φ be an LTL formula over alphabet 2^{Σ_i} . Let $\mathcal{A}_{\varphi} = (Q, q_0, \delta, F)$ be an ACA with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_{\varphi})$. Let $\mathcal{A}_{\neg\varphi} = (Q^c, q_0^c, \delta^c, F^c)$ be an ACA with $\mathcal{L}(\neg\varphi) = \mathcal{L}(\mathcal{A}_{\neg\varphi})$. We construct the ACA $\mathcal{B}_{\mathcal{A}_{\varphi}}^A = (Q^A, Q_0^A, \delta^A, F^A)$ with alphabet $2^{\Sigma_i \cup O'_i}$ as follows.

- $Q^A := (Q \times Q \times \{\top, \perp\}) \cup Q^c$
- $Q_0^A := (q_0, q_0, \top)$
- $F^A := (Q \times Q \times \{\perp\}) \cup F^c$
- $\delta^A : ((Q \times Q \times \{\top, \perp\}) \cup Q^c) \times 2^{\Sigma_i \cup O'_i} \rightarrow (Q \times Q \times \{\top, \perp\}) \cup Q^c$ with

$$\begin{aligned} \delta^A(q_c, \tilde{\iota}) &:= \delta^c(q_c, \iota') \quad \text{for } q_c \in Q^c \\ \delta^A((q_0, q_0, \top), \tilde{\iota}) &:= \delta^c(q_0, \iota') \vee \bigwedge_{c \in \delta(q_0, \iota')} \bigvee_{c' \in \delta(q_0, \iota)} \bigwedge_{q' \in c'} \bigvee_{p' \in c} \vartheta(p', q', \top) \\ \delta^A((p, q, m), \tilde{\iota}) &:= \bigwedge_{c \in \delta(p, \iota')} \bigvee_{c' \in \delta(q, \iota)} \bigwedge_{q' \in c'} \bigvee_{p' \in c} \vartheta(p', q', m) \end{aligned}$$

where $\iota := \tilde{\iota} \cap \Sigma_i$, $\iota' := unpr(\tilde{\iota} \cap \Sigma'_i)$, and $\vartheta : (Q \times Q \times \{\top, \perp\}) \rightarrow Q \times Q \times \{\top, \perp\}$ with

$$\vartheta(p, q, m) := \begin{cases} (p, q, \perp) & \text{if } p \notin F, q \in F, \text{ and } m = \top \\ (p, q, \perp) & \text{if } p \notin F \text{ and } m = \perp \\ (p, q, \top) & \text{otherwise} \end{cases}$$

Note that $\mathcal{B}_{\mathcal{A}_\varphi}^A$ indeed consists of two parts: the one defined by states of the form (p, q, m) , and the one defined by the states of $\mathcal{A}_{\neg\varphi}$. By definition of δ^A , these parts are only connected in the initial state of $\mathcal{B}_{\mathcal{A}_\varphi}^A$, where a nondeterministic transition to the respective successors in both parts ensures that choosing nondeterministically whether (i) or (ii) will be satisfied is possible. For states of the form (p, q, m) , the mark $m \in \{\top, \perp\}$ determines whether there are *pending visits to rejecting states* in the copy of \mathcal{A}_φ for the dominant strategy, i.e., the second component q of (p, q, m) . A pending visit to a rejecting state is one that is not yet matched by a visit to a rejecting state in the copy of \mathcal{A}_φ for the alternative strategy. Thus, ϑ defines that if a visit to a rejecting dominant state, that is not immediately matched with a rejecting alternative state, is encountered, the mark is set to \perp . As long as no rejecting alternative state is visited, the mark stays set to \perp . If a matching rejecting alternative state occurs, however, the mark is reset to \top . States of $\mathcal{B}_{\mathcal{A}_\varphi}^A$ marked with \perp are then defined to be rejecting states, ensuring that a visit to a rejecting dominant state is not pending forever.

The ACA $\mathcal{B}_{\mathcal{A}_\varphi}^A$ constructed from ACAs \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$ according to Definition 10 is sound and complete in the sense that it recognizes whether or not a strategy s delay-dominates another strategy t on an input sequence $\gamma \in (2^{I_i})^\omega$. That is, $\mathcal{B}_{\mathcal{A}_\varphi}^A$ accepts the infinite word $\text{comp}(s, \gamma) \cup \text{pr}(\text{comp}(t, \gamma) \cap O_i)$ if, and only if, $t \leq_{\gamma}^{\mathcal{A}_\varphi} s$ holds for \mathcal{A}_φ . The main idea is that a run tree of $\mathcal{B}_{\mathcal{A}_\varphi}^A$ can be translated into a strategy for Duplicator in the delay-dominance game and vice versa since, by construction, both define the existential choices in \mathcal{A}_φ for s and the universal choices in \mathcal{A}_φ for t . Thus, for a run tree of $\mathcal{B}_{\mathcal{A}_\varphi}^A$ whose branches all visit only finitely many rejecting states, there exists a strategy for Duplicator in the delay-dominance game that ensures that for all consistent plays either $\text{comp}(t, \gamma) \not\models \varphi$ holds or, by construction of ϑ and δ^A , every rejecting dominant state is matched by a rejecting alternative state eventually. Similarly, a winning strategy for Duplicator can be translated into a run tree r of $\mathcal{B}_{\mathcal{A}_\varphi}^A$. If $\text{comp}(t, \gamma) \models \varphi$ holds, then r visits only finitely many rejecting states since only finitely many rejecting dominant states are visited. If $\text{comp}(t, \gamma) \not\models \varphi$ holds, then there exists a run tree, namely one entering the part of $\mathcal{B}_{\mathcal{A}_\varphi}^A$ that coincides with $\mathcal{A}_{\neg\varphi}$, whose branches all visit only finitely many rejecting states. The proof is given in [19].

► **Lemma 11.** *Let φ be an LTL formula. Let \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$ be ACAs with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$ and $\mathcal{L}(\neg\varphi) = \mathcal{L}(\mathcal{A}_{\neg\varphi})$. Let $\mathcal{B}_{\mathcal{A}_\varphi}^A$ be the ACA constructed from \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$ according to Definition 10. Let s and t be strategies for process p_i . Let $\gamma \in (2^{I_i})^\omega$. Let $\sigma \in (2^{\Sigma_i \cup O'_i})^\omega$ with $\sigma := \text{comp}(s, \gamma) \cup \text{pr}(\text{comp}(t, \gamma) \cap O_i)$. Then, $\mathcal{B}_{\mathcal{A}_\varphi}^A$ accepts σ if, and only if, $t \leq_{\gamma}^{\mathcal{A}_\varphi} s$ holds.*

Thus, $\mathcal{B}_{\mathcal{A}_\varphi}^A$ determines whether or not a strategy s delay-dominates a strategy t . However, $\mathcal{B}_{\mathcal{A}_\varphi}^A$ cannot directly be used for synthesizing delay-dominant strategies since (i) $\mathcal{B}_{\mathcal{A}_\varphi}^A$ is an alternating automaton, while we require a universal automaton for bounded synthesis, and (ii) $\mathcal{B}_{\mathcal{A}_\varphi}^A$ considers *one particular* alternative strategy t . For recognizing delay-dominance, we need to consider *all* alternative strategies, though. Thus, we describe in the remainder of this section how $\mathcal{B}_{\mathcal{A}_\varphi}^A$ can be translated into a UCA for bounded synthesis.

5.2 Construction of the UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$

Next, we translate the ACA $\mathcal{B}_{\mathcal{A}_\varphi}^A$ constructed in the previous subsection to a UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ that can be used for synthesizing delay-dominant strategies. As outlined before, we need to (i) translate $\mathcal{B}_{\mathcal{A}_\varphi}^A$ into a UCA, and (ii) ensure that the automaton considers *all* alternative strategies instead of a particular one. Thus, we proceed in two steps. First, we translate $\mathcal{B}_{\mathcal{A}_\varphi}^A$ into an equivalent UCA $\mathcal{B}_{\mathcal{A}_\varphi}^U$. We utilize the Miyano-Hayashi algorithm [30] for translating ABAs into NBAs. Since we are considering co-Büchi automata instead of Büchi automata,

we further make use of the duality of nondeterministic and universal branching and the Büchi and co-Büchi acceptance conditions. The translation introduces an exponential blow-up in the number of states. For the full construction, we refer to [19].

► **Lemma 12.** *Let \mathcal{A} be an alternating co-Büchi automaton with m states. There exists a universal co-Büchi automaton \mathcal{B} with $\mathcal{O}(2^m)$ states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ holds.*

Next, we construct the desired universal co-Büchi automaton $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ that recognizes delay-dominant strategies for \mathcal{A}_φ . For this sake, we need to adapt $\mathcal{B}_{\mathcal{A}_\varphi}^U$ to consider *all* alternative strategies instead of a particular one. Similar to the automaton construction for synthesizing remorsefree dominant strategies [10, 16], we utilize *universal projection*:

► **Definition 13 (Universal Projection).** *Let $\mathcal{A} = (Q, Q_0, \delta, F)$ be a UCA over alphabet Σ and let $X \subset \Sigma$. The universal projection of \mathcal{A} to X is the UCA $\pi_X(\mathcal{A}) = (Q, Q_0, \pi_X(\delta), F)$ over alphabet X , where $\pi_X(\delta) = \{(q, a, q') \in Q \times 2^X \times Q \mid \exists b \in 2^{\Sigma \setminus X}. (q, a \cup b, q') \in \delta\}$.*

The projected automaton $\pi_X(\mathcal{A})$ for a UCA \mathcal{A} over Σ and a set $X \subset \Sigma$ contains the transitions of \mathcal{A} for *all* possible valuations of the variables in $\Sigma \setminus X$. Hence, for a sequence $\sigma \in (2^X)^\omega$, all runs of \mathcal{A} on sequences extending σ with some valuation of the variables in $\Sigma \setminus X$ are also runs of $\pi_X(\mathcal{A})$. Since both \mathcal{A} and $\pi_X(\mathcal{A})$ are universal automata, $\pi_X(\mathcal{A})$ thus accepts a sequence $\sigma \in (2^X)^\omega$ if, and only if, \mathcal{A} accepts all sequences extending σ with some valuation of the variables in $\Sigma \setminus X$. The proof is given in [19].

► **Lemma 14.** *Let \mathcal{A} be a UCA over alphabet Σ and let $X \subset \Sigma$. Let $\sigma \in (2^X)^\omega$. Then, $\pi_X(\mathcal{A})$ accepts σ if, and only if \mathcal{A} accepts all $\sigma' \in (2^\Sigma)^\omega$ with $\sigma' \cap X = \sigma$.*

We utilize this property to obtain a universal co-Büchi automaton $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ from $\mathcal{B}_{\mathcal{A}_\varphi}^U$ that considers *all* possible alternative strategies instead of only a particular one: we project to the unprimed variables of p_i , i.e., to Σ_i , thereby quantifying universally over the alternative strategies. We thus obtain a UCA that recognizes delay-dominant strategies as follows:

► **Definition 15 (Delay-Dominance Automaton).** *Let φ be an LTL formula. Let $\mathcal{A}_\varphi, \mathcal{A}_{\neg\varphi}$ be ACAs with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$, $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$. Let $\mathcal{B}_{\mathcal{A}_\varphi}^A$ be the ACA constructed from \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$ according to Definition 10. Let $\mathcal{B}_{\mathcal{A}_\varphi}^U$ be a UCA with $\mathcal{L}(\mathcal{B}_{\mathcal{A}_\varphi}^A) = \mathcal{L}(\mathcal{B}_{\mathcal{A}_\varphi}^U)$. The delay-dominance UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ for \mathcal{A}_φ and process p_i is then given by $\mathcal{A}_{\mathcal{A}_\varphi}^{dd} := \pi_{\Sigma_i}(\mathcal{B}_{\mathcal{A}_\varphi}^U)$.*

Utilizing the previous results, we can now show soundness and completeness of the delay-dominance universal co-Büchi automaton $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$: from Lemma 11, we know that $\mathcal{B}_{\mathcal{A}_\varphi}^A$ recognizes whether or not a strategy s for a process p_i delay-dominates another strategy t for p_i for \mathcal{A}_φ on an input sequence $\gamma \in (2^{I_i})^\omega$. By Lemma 12, we have $\mathcal{L}(\mathcal{B}_{\mathcal{A}_\varphi}^U) = \mathcal{L}(\mathcal{B}_{\mathcal{A}_\varphi}^A)$. With the definition of the delay-dominance UCA, namely $\mathcal{A}_{\mathcal{A}_\varphi}^{dd} := \pi_{\Sigma_i}(\mathcal{B}_{\mathcal{A}_\varphi}^U)$, as well as with Lemma 14, it then follows that $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ accepts $\text{comp}(s, \gamma)$ for all input sequences $\gamma \in (2^{I_i})^\omega$ if, and only if, s is delay-dominant for \mathcal{A}_φ . For the formal proof, we refer to [19].

► **Theorem 16 (Soundness and Completeness).** *Let φ be an LTL formula and let \mathcal{A}_φ be an ACA with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. Let $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ be the delay-dominance UCA for \mathcal{A}_φ as constructed in Definition 15. Let s be a process strategy for process p_i . Then $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ accepts $\text{comp}(s, \gamma)$ for all input sequences $\gamma \in (2^{I_i})^\omega$, if, and only if s is delay-dominant for \mathcal{A}_φ and p_i .*

Furthermore, $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ is of convenient size: for an LTL formula φ , there is an ACA \mathcal{A}_φ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ such that $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ constructed from \mathcal{A}_φ is of exponential size in the squared length of the formula φ . This follows from Lemma 12 and from the facts that (i) \mathcal{A}_φ and $\mathcal{A}_{\neg\varphi}$ both are of linear size in the length of the LTL formula φ , and (ii) universal projection preserves the automaton size. The proof is given in [19].

► **Lemma 17.** *Let φ be an LTL formula and let s be a strategy for process p_i . There is an ACA \mathcal{A}_φ of size $\mathcal{O}(|\varphi|)$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ and a UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ of size $\mathcal{O}(2^{|\varphi|^2})$ such that $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ accepts $\text{comp}(s, \gamma)$ for all $\gamma \in (2^I)^\omega$, if, and only if, s is delay-dominant for \mathcal{A}_φ and p_i .*

Since the automaton construction described in this section is sound and complete, the UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ can be used for synthesizing delay-dominant strategies. In fact, it immediately enables utilizing existing bounded synthesis tools for the synthesis of delay-dominant strategies by replacing the UCA recognizing winning strategies with $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$.

Note that, similar as for the UCA recognizing remorsefree dominance [10], $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ can be translated into a nondeterministic parity tree automaton with an exponential number of colors and a doubly-exponential number of states in the squared length of the formula. Synthesizing delay-dominant strategies thus reduces to checking tree automata emptiness and, if the automaton is non-empty, to extracting a Moore machine representing a process strategy from an accepted tree. This can be done in exponential time in the number of colors and in polynomial time in the number of states [25]. With Lemma 17, a doubly-exponential complexity for synthesizing delay-dominant strategies thus follows:

► **Theorem 18.** *Let φ be an LTL formula and let \mathcal{A}_φ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. If there exists a delay-dominant strategy for \mathcal{A}_φ , then it can be computed in 2EXPTIME.*

It is well-known that synthesizing winning strategies is 2EXPTIME-complete [32]. Since there exists a UCA of exponential size in the length of the formula which recognizes remorsefree dominant strategies, dominant strategies can also be synthesized in 2EXPTIME [10]. Synthesizing delay-dominant strategies rather than winning or remorsefree dominant ones thus does not introduce any overhead, while it allows for a simple compositional synthesis approach for distributed systems for many safety and liveness specifications.

6 Compositional Synthesis with Delay-Dominant Strategies

In this section, we describe a compositional synthesis approach that utilizes delay-dominant strategies. We extend the algorithm described in [10] from safety specifications to general properties by synthesizing delay-dominant strategies instead of remorsefree dominant ones. Hence, given a distributed architecture and an LTL specification φ , the compositional synthesis algorithm proceeds in four steps. First, φ is translated into an equivalent ACA \mathcal{A}_φ using standard algorithms. Second, for each system process p_i , we construct the UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ that recognizes delay-dominant strategies for φ and p_i as described in Section 5. Note that although the initial automaton \mathcal{A}_φ is the same for every process p_i , the UCAs recognizing delay-dominant strategies differ: since the processes have different sets of output variables, already the alphabets of the intermediate ACA $\mathcal{B}_{\mathcal{A}_\varphi}^A$ differ for different processes. Third, a delay-dominant strategy s_i is synthesized for each process p_i from the respective UCA $\mathcal{A}_{\mathcal{A}_\varphi}^{dd}$ with bounded synthesis. Lastly, the strategies s_1, \dots, s_n are composed according to the definition of the parallel composition of Moore machines (see Section 2) into a single strategy s for the whole distributed system. By Theorem 8, the composed strategy s is delay-dominant for \mathcal{A}_φ and the whole system if \mathcal{A}_φ ensures bad prefixes for delay-dominance. If φ is realizable, then, by Lemma 6, strategy s is guaranteed to be winning for φ .

Note that even for realizable LTL formulas φ , there does not necessarily exist a delay-dominant strategy since delay-dominance is not solely defined on the satisfaction of φ but on the *structure* of an equivalent ACA \mathcal{A}_φ . In certain cases, \mathcal{A}_φ can thus “punish” the delay-dominant strategy by introducing rejecting states at clever positions that do not influence acceptance but delay-dominance, preventing the existence of a delay-dominant strategy.

However, we experienced that an ACA \mathcal{A}_φ constructed with standard algorithms from an LTL formula φ does not punish delay-dominant strategies since \mathcal{A}_φ thoroughly follows the structure of φ and thus oftentimes does not contain unnecessary rejecting states. Furthermore, such an ACA oftentimes ensure bad prefixes for delay-dominance: in [19], we discuss under which circumstances the bad prefix property is not satisfied and identify critical structures in co-Büchi automata. When constructing ACAs with standard algorithms from LTL formulas, such structures rarely – if ever – exist. Simple optimizations like removing rejecting states that do not lie in a cycle from the set of rejecting states have a positive impact on both the existence of delay-dominant strategies and on ensuring bad prefixes: such states cannot be visited infinitely often and thus removing them from the set of rejecting states does not alter the language. Nevertheless, rejecting states can enforce non-delay-dominance and thus removing unnecessary rejecting states can result in more strategies being delay-dominant. Note that with this optimization it, for instance, immediately follows that for safety properties the parallel composition of delay-dominant strategies is delay-dominant. Thus, we experienced that for an ACA \mathcal{A}_φ constructed from an LTL formula φ with standard algorithms it holds in many cases that (i) if φ allows for a remorsefree dominant strategy, then \mathcal{A}_φ allows for an delay-dominant strategy, and (ii) the parallel composition of delay-dominance strategies for \mathcal{A}_φ is delay-dominant as well. Therefore, the compositional synthesis algorithm presented in this section is indeed applicable for many LTL formulas.

7 Conclusion

We have presented a new winning condition for process strategies, delay-dominance, that allows a strategy to violate a given specification in certain situations. In contrast to the classical notion of winning, delay-dominance can thus be used for individually synthesizing strategies for the processes in a distributed system in many cases, therefore enabling a simple compositional synthesis approach. Delay-dominance builds upon remorsefree dominance, where a strategy is allowed to violate the specification as long as no other strategy would have satisfied it in the same situation. However, remorsefree dominance is only compositional for safety properties. For liveness properties, the parallel composition of dominant strategies is not necessarily dominant. This restricts the use of dominance-based compositional synthesis algorithms to safety specifications, which are often not expressive enough. Delay-dominance, in contrast, is specifically designed to be compositional for more properties. We have introduced a game-based definition of delay-dominance as well as a criterion such that, if the criterion is satisfied, compositionality of delay-dominance is guaranteed; both for safety and liveness properties. Furthermore, every delay-dominant strategy is remorsefree dominant, and, for realizable system specifications, the parallel composition of delay-dominant strategies for all system processes is guaranteed to be winning for the whole system if the criterion is satisfied. Hence, delay-dominance is a suitable notion for compositional synthesis algorithms. We have introduced an automaton construction for recognizing delay-dominant strategies. The resulting universal co-Büchi automaton can immediately be used to synthesize delay-dominant strategies utilizing existing bounded synthesis approaches. The automaton is of single-exponential size in the squared length of the specification. Thus, synthesizing delay-dominant strategies is, as for winning and remorsefree ones, in 2EXPTIME.

References

- 1 Shaull Almagor and Orna Kupferman. Good-Enough Synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification – 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

- 2 Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Synthesis under Assumptions. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October – 2 November 2018*, pages 615–616. AAAI Press, 2018. URL: <https://aaai.org/ocs/index.php/KR/KR18/paper/view/18053>.
- 3 Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Best-Effort Synthesis: Doing Your Best is Not Harder Than Giving Up. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1766–1772. ijcai.org, 2021. doi:10.24963/ijcai.2021/243.
- 4 Jan E. Baumeister. Encodings of Bounded Synthesis for Distributed Systems. Bachelor’s Thesis, Saarland University, 2017.
- 5 Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to Handle Assumptions in Synthesis. In Krishnendu Chatterjee, Rüdiger Ehlers, and Susmit Jha, editors, *Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014*, volume 157 of *EPTCS*, pages 34–50, 2014. doi:10.4204/EPTCS.157.7.
- 6 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-Admissible Synthesis. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015*, volume 42 of *LIPICs*, pages 100–113. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CONCUR.2015.100.
- 7 Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment Assumptions for Synthesis. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 – Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008. doi:10.1007/978-3-540-85361-9_14.
- 8 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The Complexity of Rational Synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.121.
- 9 Werner Damm and Bernd Finkbeiner. Does It Pay to Extend the Perimeter of a World Model? In Michael J. Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods – 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2011. doi:10.1007/978-3-642-21437-0_4.
- 10 Werner Damm and Bernd Finkbeiner. Automatic Compositional Synthesis of Distributed Systems. In *FM 2014: Formal Methods – 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2014. doi:10.1007/978-3-319-06410-9_13.
- 11 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of Bounded Synthesis. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017. doi:10.1007/978-3-662-54577-5_20.
- 12 Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. BoSy: An Experimentation Framework for Bounded Synthesis. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification – 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 325–332. Springer, 2017. doi:10.1007/978-3-319-63390-9_17.

- 13 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Compositional Algorithms for LTL Synthesis. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *Automated Technology for Verification and Analysis – 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*, volume 6252 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2010. doi:10.1007/978-3-642-15643-4_10.
- 14 Bernd Finkbeiner, Gideon Geier, and Noemi Passing. Specification Decomposition for Reactive Synthesis. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods – 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings*, volume 12673 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2021. doi:10.1007/978-3-030-76384-8_8.
- 15 Bernd Finkbeiner, Gideon Geier, and Noemi Passing. Specification decomposition for reactive synthesis. *Innovations Syst. Softw. Eng.*, 2022. doi:10.1007/s11334-022-00462-6.
- 16 Bernd Finkbeiner and Noemi Passing. Dependency-Based Compositional Synthesis. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis – 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 2020. doi:10.1007/978-3-030-59152-6_25.
- 17 Bernd Finkbeiner and Noemi Passing. Compositional Synthesis of Modular Systems. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis – 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 303–319. Springer, 2021. doi:10.1007/978-3-030-88885-5_20.
- 18 Bernd Finkbeiner and Noemi Passing. Compositional synthesis of modular systems. *Innov. Syst. Softw. Eng.*, 18(3):455–469, 2022. doi:10.1007/s11334-022-00450-w.
- 19 Bernd Finkbeiner and Noemi Passing. Synthesizing Dominant Strategies for Liveness (Full Version). *CoRR*, abs/2210.01660, 2022. doi:10.48550/arXiv.2210.01660.
- 20 Bernd Finkbeiner and Sven Schewe. Uniform Distributed Synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 321–330. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.53.
- 21 Bernd Finkbeiner and Sven Schewe. SMT-Based Synthesis of Distributed Systems. In *Proc. AFM*, 2007.
- 22 Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013. doi:10.1007/s10009-012-0228-z.
- 23 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational Synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.
- 24 Carsten Fritz and Thomas Wilke. Simulation Relations for Alternating Büchi Automata. *Theor. Comput. Sci.*, 338(1-3):275–314, 2005. doi:10.1016/j.tcs.2005.01.016.
- 25 Marcin Jurdzinski. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.
- 26 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with Rational Environments. In Nils Bulling, editor, *Multi-Agent Systems – 12th European Conference, EUMAS 2014, Prague, Czech Republic, December 18-19, 2014, Revised Selected Papers*, volume 8953 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2014. doi:10.1007/978-3-319-17130-2_15.
- 27 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safrless Compositional Synthesis. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International*

- Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2006. doi:10.1007/11817963_6.
- 28 Orna Kupferman and Moshe Y. Vardi. Safrless Decision Procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 531–542. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.66.
- 29 Yong Li, Andrea Turrini, Moshe Y. Vardi, and Lijun Zhang. Synthesizing Good-Enough Strategies for LTLf Specifications. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4144–4151. ijcai.org, 2021. doi:10.24963/ijcai.2021/570.
- 30 Satoru Miyano and Takeshi Hayashi. Alternating Finite Automata on ω -Words. *Theor. Comput. Sci.*, 32:321–330, 1984. doi:10.1016/0304-3975(84)90049-5.
- 31 Amir Pnueli. The Temporal Logic of Programs. In *Annual Symposium on Foundations of Computer Science, 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 32 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.
- 33 Amir Pnueli and Roni Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/SFCS.1990.89597.

Low-Latency Sliding Window Algorithms for Formal Languages

Moses Ganardi  


Max Planck Institute for Software Systems, Kaiserslautern, Germany

Louis Jachiet  

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Markus Lohrey  

Universität Siegen, Germany

Thomas Schwentick  

TU Dortmund University, Germany

Abstract

Low-latency sliding window algorithms for regular and context-free languages are studied, where latency refers to the worst-case time spent for a single window update or query. For every regular language L it is shown that there exists a constant-latency solution that supports adding and removing symbols independently on both ends of the window (the so-called two-way variable-size model). We prove that this result extends to all visibly pushdown languages. For deterministic 1-counter languages we present a $\mathcal{O}(\log n)$ latency sliding window algorithm for the two-way variable-size model where n refers to the window size. We complement these results with a conditional lower bound: there exists a fixed real-time deterministic context-free language L such that, assuming the OMV (online matrix vector multiplication) conjecture, there is no sliding window algorithm for L with latency $n^{1/2-\epsilon}$ for any $\epsilon > 0$, even in the most restricted sliding window model (one-way fixed-size model). The above mentioned results all refer to the unit-cost RAM model with logarithmic word size. For regular languages we also present a refined picture using word sizes $\mathcal{O}(1)$, $\mathcal{O}(\log \log n)$, and $\mathcal{O}(\log n)$.

2012 ACM Subject Classification Theory of computation \rightarrow Regular languages; Theory of computation \rightarrow Grammars and context-free languages; Theory of computation \rightarrow Streaming models

Keywords and phrases Streaming algorithms, regular languages, context-free languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.38

Related Version *Full Version:* <https://arxiv.org/abs/2209.14835>

Funding *Markus Lohrey:* Partly supported by the DFG project LO748/13-1.

1 Introduction

In this paper, we investigate sliding window algorithms for formal languages. In the basic sliding window model, an infinite stream $s = a_1a_2a_3 \dots$ of symbols from a finite alphabet Σ is read symbol by symbol from left to right. It works *one-way* and with a *fixed window size* n . The *window content* is the suffix of length n of the prefix of the stream s seen so far. Thus, in each step, a new right-most symbol is read into the window and the left-most symbol is moved out. A sliding window algorithm for a language $L \subseteq \Sigma^*$ has to indicate at every time instant whether the current window content belongs to L (initially the window is filled with some dummy symbol). The two resources that one typically tries to minimize are memory and the worst-case time spent per incoming symbol. It is important to note that the model only has access to the letter currently read. In particular, if the algorithm wants to know the precise content of the current window or, in particular, which letter moves out of the window, it has to dedicate memory for it. For general background on sliding window algorithms see [1, 9].



© Moses Ganardi, Louis Jachiet, Markus Lohrey, and Thomas Schwentick; licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 38; pp. 38:1–38:23



Leibniz International Proceedings in Informatics

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

We refer to the above sliding window model as the *one-way, fixed-size model*. A more general variant is the one-way, *variable-size* sliding window model. In this model the arrival of new symbols and the expiration of old symbols are handled independently, i.e., there are update operations that add a new right-most symbol and an operation that removes the left-most symbol. Therefore the size of the window can grow and shrink. This allows to model for instance a time-based window that contains all data values that have arrived in the last t seconds for some fixed t . If the arrival times are arbitrary then the window size may vary. The *two-way model* is a further generalization whose update operations allow to add and remove symbols on both sides of the window. It can be combined with both the fixed-size and the variable-size model. The variable-size two-way model is the most general model; it is also known as a deque (double-ended queue); see [25, Section 2.2.1]. An algorithm also needs to handle query operations, asking whether the current window content is in the language L .

There are two important complexity measures for a sliding window algorithm: its *space complexity* and its *latency* (or *time complexity*), i.e. the time required for a single update or query operation. Both are usually expressed depending on the window size n (for a fixed-size sliding window algorithm) or the maximal window size n that occurs during a run of the sliding window algorithm (for a variable-size sliding window algorithm). In this paper we are mainly interested in the *latency* of sliding window algorithms. Since it turns out that space complexity is an important tool for proving lower bounds on the latency of sliding window algorithms, we first discuss known results on space complexity.

Space complexity of sliding window algorithms. The space complexity of sliding window algorithms for formal languages in the one-way (fixed-size and variable-size) model has been studied in [15, 16, 17, 19, 21] and in [18, Section 9] for the two-way variable-size model. For regular languages, the main result of [17] is a space trichotomy for the one-way case: the space complexity of a regular language is either constant, logarithmic or linear. This result holds for the fixed-size model as well as the variable-size model, although the respective language classes differ slightly. For the two-way variable-size model a space trichotomy has been shown in [18]. Table 1 summarizes some of the main results of [16, 17, 18] in more detail (ignore the word size bound for the moment). The results on the two-way fixed-size model in Table 1 are shown in this paper. In that table,

- Reg denotes the class of all regular languages;
- Len denotes the class of *regular length languages*, i.e., regular languages $L \subseteq \Sigma^*$, for which either $\Sigma^n \subseteq L$ or $\Sigma^n \cap L = \emptyset$, for every n ;
- LI denotes the class of all *regular left ideals*, i.e., regular languages of the form Σ^*L for a regular language L ;
- SL denotes the class of *suffix languages*¹, i.e., languages of the form Σ^*w ;
- Triv denotes class of trivial languages, i.e., the class consisting of \emptyset and Σ^* only;
- Fin denotes the class of finite languages;
- $\langle A_1, \dots, A_n \rangle$ denotes the Boolean closure of $\bigcup_{1 \leq i \leq n} A_i$.

Note that these classes are defined with respect to an alphabet, e.g. a^* is trivial if the alphabet is $\{a\}$ but non-trivial if the alphabet is $\{a, b\}$. In Table 1, we write $f \in \bar{\Theta}(g)$ for $f, g : \mathbb{N} \rightarrow \mathbb{R}$ iff $f \in \mathcal{O}(g)$ and there is a constant $c > 0$ with $f(n) \geq c \cdot g(n)$ for infinitely many n .

¹ In [16, 17], we used instead of SL the boolean closure of SL (the so-called suffix testable language); but this makes no difference, since we are only interested in the boolean closures of language classes.

■ **Table 1** Summary of results for regular languages and constant latency. The columns correspond to the 4 different sliding window models (1F = one-way fixed-size, 1V = one-way variable-size, 2F = two-way fixed-size, 2V = two-way variable-size). The rows correspond to different combinations of word size and space in bits. For all three combinations the latency is $\mathcal{O}(1)$. Note that the language classes in each column yield a partition of Reg .

	1F	1V	2F	2V
word size: $\mathcal{O}(1)$ space in bits: $\mathcal{O}(1)$	$\langle \text{Len}, \text{SL} \rangle$	Triv	$\langle \text{Len}, \text{Fin} \rangle$	Triv
word size: $\bar{\mathcal{O}}(\log \log n)$ space in bits: $\bar{\mathcal{O}}(\log n)$	$\langle \text{Len}, \text{LI} \rangle \setminus \langle \text{Len}, \text{SL} \rangle$	$\langle \text{Len}, \text{LI} \rangle \setminus \text{Triv}$	\emptyset	$\text{Len} \setminus \text{Triv}$
word size: $\bar{\mathcal{O}}(\log n)$ space in bits: $\bar{\mathcal{O}}(n)$	$\text{Reg} \setminus \langle \text{Len}, \text{LI} \rangle$	$\text{Reg} \setminus \langle \text{Len}, \text{LI} \rangle$	$\text{Reg} \setminus \langle \text{Len}, \text{Fin} \rangle$	$\text{Reg} \setminus \text{Len}$

Some of the results from [16, 17] were extended to (subclasses of) context-free languages in [15, 21]. A space trichotomy was shown for visibly pushdown languages in [15], whereas for the class of all deterministic context-free languages the space trichotomy fails [21].

Content of the paper. In this paper we consider the latency of sliding window algorithms for regular and deterministic context-free languages in all four of the above models: one-way and two-way, fixed-size and variable-size. These models are formally defined in Section 2. As the algorithmic model, we use the standard RAM model. The word size (register length) is a parameter in this model and we allow it to depend on the fixed window size n (in the fixed-size model) or the maximal window size n that has occurred in the past (for the variable-size model). More precisely, depending on the language class, the word size can be $\mathcal{O}(1)$ (resulting in the *bit-cost model*), $\mathcal{O}(\log \log n)$, or $\mathcal{O}(\log n)$. We assume the unit-cost measure, charging a cost of 1 for each basic register operation.

The bit-cost model serves as a link to transfer lower bounds: it is a simple observation, formalized in Lemma 1, that the sliding window time complexity for a language L in the bit-cost model is at least the logarithm of the minimum possible space complexity. In fact, this lower bound holds even with respect to the non-uniform bit-probe model, where we have a separate algorithm for each window size n . And, again by Lemma 1, lower bounds on the latency in the bit-cost model translate to lower bounds on the word size for unit-cost algorithms with constant latency.

In Section 3, we study the latency of sliding window algorithms for regular languages and offer a complete picture. Our contribution here is mainly of algorithmic nature, since most of the lower bounds are simple consequences of space lower bounds, that were shown in [16, 17, 18]. The main result of the first part is that these lower bounds can be achieved by concrete algorithms. More precisely, there are algorithms that (1) achieve the optimal latency with respect to the bit-cost model and (2) constant latency with respect to unit-cost model, and (3) also have optimal space complexity. The precise results are summarized in Table 1 for the unit-cost model. In all cases, the sliding window algorithms have constant latency. For example, languages from $\langle \text{Len}, \text{LI} \rangle$ have one-way sliding window algorithms with constant latency on unit-cost RAMs with word size $\mathcal{O}(\log \log n)$ and thus $\mathcal{O}(\log \log n)$ latency in the bit-cost model. These algorithms have space complexity $\mathcal{O}(\log n)$. Moreover, unless the language belongs to $\langle \text{Len}, \text{LI} \rangle$ (for the fixed-size model) or Triv (for the variable-size model)

these resource bounds cannot be improved. Note that, while for three of the four models there is a trichotomy, the two-way fixed-size model is an outlier: it has a dichotomy, since there are no languages of intermediate complexity.

In Section 4 we consider the latency for deterministic context-free languages (DCFL) and here the study is more of an explorative nature. Since every DCFL has a linear time parsing algorithm [24], one might hope to get also a low-latency sliding window algorithm. Our first result tempers this hope: assuming the OMV (online matrix vector multiplication) conjecture [23], we show that there exists a fixed real-time DCFL L such that no algorithm can solve the sliding window problem for L on a RAM with logarithmic word size with latency $n^{1/2-\epsilon}$ for any $\epsilon > 0$, even in the one-way fixed-size model (the most restricted model). This motivates to look for subclasses that allow more efficient sliding window algorithms. We present two results in this direction. We show that for every visibly pushdown language [2] there is a two-way variable-size sliding window algorithm on a unit-cost RAM with word size $\mathcal{O}(\log n)$ and constant latency. Visibly pushdown languages are widely used, e.g. for describing tree-structured documents and traces of recursive programs. They share many of the nice algorithmic and closure properties of regular languages. Finally, we show that for every deterministic one-counter language there is a two-way variable-size sliding window algorithm on a unit-cost RAM with word size $\mathcal{O}(\log n)$ and latency $\mathcal{O}(\log n)$.

Related work. The latency of regular languages in the sliding window model has been first studied in [28], where it was shown that in the one-way, fixed-size model, every regular language has a constant latency algorithm on a RAM with word size $\log n$ (the result is not explicitly stated in [28] but directly follows by using the main result of [28] for the transformation monoid of an automaton). Our upper bound results for general regular languages rely on this work and we extend its techniques to visibly pushdown languages.

A sliding window algorithm can be viewed as a dynamic data structure that maintains a dynamic string w (the window content) under very restricted update operations. Dynamic membership problems for more general updates that allow to change the symbol at an arbitrary position have been studied in [3, 13, 14].

Standard streaming algorithms (where the whole history and not only the last n symbols is relevant) for visibly pushdown languages (and subclasses) were studied in [4, 5, 6, 10, 12, 26, 27]. These papers investigate the space complexity of streaming. Update times of streaming algorithms for timed automata have been studied in [22].

2 Sliding window model

Throughout this paper we use $\log n$ as an abbreviation for $\lceil \log_2 n \rceil$.

Consider a function $f : \Sigma^* \rightarrow C$ for some finite alphabet Σ and some countable set C . We will view the sliding window problem for the function f as a dynamic data structure problem, where we want to maintain a word $w \in \Sigma^*$, called the *window*, which undergoes changes and admits membership queries to L . Altogether, we consider the following operations on Σ^* , where for a word $w = a_1 \cdots a_n \in \Sigma^*$ we write $|w| = n$ for its length and $w[i : j] = a_i \cdots a_j$ for the factor from position i to position j (which is ε if $i > j$).

- `rightpush(a)`: Replace w by wa .
- `leftpush(a)`: Replace w by aw .
- `leftpop()`: Replace w by $w[2 : |w|]$ (which is ε if $w = \varepsilon$).
- `rightpop()`: Replace w by $w[1 : |w| - 1]$ (which again is ε if $w = \varepsilon$).
- `query()`: Return the value $f(w)$.

In most cases, the function f will be the characteristic function of a language $L \subseteq \Sigma^*$; in this case we speak of the sliding window problem for the language L .

In the *two-way model* all five operations are allowed, whereas in the *one-way model*, we only allow to add symbols on the right and to remove symbols on the left, that is, it allows only the operations `rightpush(a)`, `leftpop()`, and `query()`. In the *variable-size window model* the operations can be applied in arbitrary order, but in the *fixed-size window model*, push operations always need to be followed directly by a pop operation on the other side. More formally, each `rightpush(a)` needs to be immediately followed by a `leftpop()` and (in the two-way model), each `leftpush(a)` needs to be immediately followed by a `rightpop()`. In particular, no query can occur between a `leftpush(a)` and the subsequent `rightpop()`. Therefore, as the name suggests, in the fixed-size model the string always has the same length n , for some n , if we consider a push and its successive pop operation as one operation.

In the variable-size model the window w is initially ε , whereas in the fixed-size model it is initialized as $w = \square^n$ for some default symbol $\square \in \Sigma$. We allow algorithms a preprocessing phase and disregard the time they spend during this initialization. In the fixed-size model the algorithm receives the window size n for its initialization.

We denote the four combinations of models by 1F, 1V, 2F, and 2V, where 1 and 2 refer to one-way and two-way, respectively, and F and V to fixed-size and variable-size respectively.

We use two different computational models to present our results, the uniform *word RAM model* for upper bounds and the non-uniform *cell probe model* for lower bounds.

Word RAM model. We present algorithmic results (i.e., upper bounds) in the *word RAM model* with maximal word size (or register length) of $b(n)$ bits, for some *word size function* $b(n)$. Algorithms may also use registers of length smaller than $b(n)$, for a more fine-grained analysis. In the fixed-size model n is the fixed window size, whereas in the variable-size model n is the maximum window size that has appeared in the past. In particular, if the window size increases then also the allowed word size $b(n)$ increases, whereas a subsequent reduction of the window size does not decrease the allowed word size. As usual, all RAM-operations on registers of word size at most $b(n)$ take constant time (unit-cost assumption). For a model $M \in \{1F, 1V, 2F, 2V\}$, an M -algorithm (for a function f) is a sliding window algorithm that supports the operations of model M .

An M -algorithm \mathcal{A} has *latency* (or *time complexity*) $T(n)$ if for all n the following hold:

- If $M \in \{1F, 2F\}$, then in every computation of window size n , all operations of model M are handled within $T(n)$ steps by \mathcal{A} .
- If $M \in \{1V, 2V\}$, then in every computation of maximal window size n , all operations of model M are handled within $T(n)$ steps by \mathcal{A} .

Space complexity is defined accordingly and refers to the number of bits used by the algorithm.

Cell probe model. For lower bounds we use the *cell probe model*. We formalize the model only for sliding window algorithms. For a model $M \in \{1F, 1V, 2F, 2V\}$, an M -algorithm in the cell probe model is a collection $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$, where \mathcal{A}_n is an M -algorithm for window size n (if $M = 1F$ or $M = 2F$), respectively, maximal window size n (if $M = 1V$ or $M = 2V$). Furthermore, we only count the number of read/write accesses to memory cells and disregard computation completely. We also say that $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$ is a *non-uniform M -algorithm*.

More formally, fix a word size function $b = b(n)$. In the cell probe model an M -algorithm \mathcal{A}_n for (maximal) window length n is a collection of decision trees $t_{n, \text{op}}$ for every operation `op` of model M . Each node of $t_{n, \text{op}}$ is labelled with a register operation `read(R_i)` or `write(R_i, u)` where i is a register address and $u \in \{0, 1\}^{b_i}$. Here, $b_i \leq b(n)$ denotes the size (in bits) of register i . A node labelled with `read(R_i)` has 2^{b_i} children, one for each possible value of register i . A node labelled with `write(R_i, w)` has exactly one child. Moreover, the leaves of

the decision tree for `query()` are labelled with output values (0 or 1). The latency $T^{\mathcal{A}}(n)$ of \mathcal{A} is the maximal height of a decision tree $t_{n,\text{op}}$. The space complexity $S^{\mathcal{A}}(n)$ of \mathcal{A} is the sum over the bit lengths of the different registers referenced in all trees $t_{n,\text{op}}$.

By minimizing for every n the number of bits used in the trees $t_{n,\text{op}}$, it follows that for every language $L \subseteq \Sigma^*$ there is a (non-uniform) M -algorithm \mathcal{B} with optimal space complexity $S^{\mathcal{B}}(n)$ for every n . We denote this optimal space complexity by $S_L^M(n)$; see also [16]. Since for every language $L \subseteq \Sigma^*$ there is a non-uniform M -algorithm that stores the window explicitly with $n \cdot \log |\Sigma|$ bits, it holds $S_L^M(n) \leq n \cdot \log |\Sigma|$.

Space complexity in the sliding window model was analyzed in [15, 16, 17, 19, 21] for the one-sided models (1F and 1V) and [18, Section 9] for the model 2V (the model 2F has not been studied so far). In these papers, the space complexity was defined slightly different but equivalent to our definition. Note that lower bounds (for space and time) that are proved for the cell probe model also hold for the (uniform) RAM model.

As mentioned before, we will use the non-uniform cell probe model only for lower bounds. Lower bounds on the space complexity of sliding window algorithms yield lower bounds on the latency, as well. In fact, all our (unconditional) lower bounds stem from space lower bounds with the help of the following lemma; shown in the long version [20]. We mainly apply space lower bounds from [16, 17, 18].

► **Lemma 1.** *For each model $M \in \{1F, 1V, 2F, 2V\}$ and each non-uniform M -algorithm \mathcal{A} with word size $b(n)$ for some language L , it holds $b(n) \cdot T^{\mathcal{A}}(n) \geq \log S_L^M(n) - \mathcal{O}(1)$.*

3 Regular languages

As mentioned in the introduction, the class of regular languages satisfies a space trichotomy in the models 1F and 1V [16, 17]: a regular language either has space complexity $\bar{\Theta}(n)$ or $\bar{\Theta}(\log n)$ or $\mathcal{O}(1)$. In the light of Lemma 1, the best we can therefore hope for are constant latency sliding-window algorithms with word size $\mathcal{O}(\log n)$, $\mathcal{O}(\log \log n)$ and $\mathcal{O}(1)$, respectively. It turns out that such algorithms actually exist. In the following, we consider each of these three levels separately. We present algorithms and confirm their optimality by corresponding lower bounds.

Before we start, let us fix our (standard) notation for finite automata. A *deterministic finite automaton* (DFA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ where Q is a finite set of states, Σ is an alphabet, $q_0 \in Q$ is the initial state, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function and $F \subseteq Q$ is the set of final states. The transition function δ is extended to a function $\delta: Q \times \Sigma^* \rightarrow Q$ in the usual way. The language accepted by \mathcal{A} is denoted by $L(\mathcal{A})$.

3.1 Logarithmic word size

For the upper bound, we show that regular languages have constant latency sliding-window algorithms with logarithmic word size and optimal space complexity in the two-way variable-size model and thus in all four models.

To this end, we start from a known 1V-algorithm for evaluating products over finite monoids and adapt it so that it also works for the two-way model and meets our optimality requirements. Recall that a *monoid* is a set \mathcal{M} equipped with an associative binary operation on \mathcal{M} . Let $\text{prod}_{\mathcal{M}}: \mathcal{M}^* \rightarrow \mathcal{M}$ be the function which maps a word over \mathcal{M} to its product. There is a folklore simulation of a queue (aka. 1V-sliding window) by two stacks, which takes constant time per operation on *average*, see [28]. This idea can be turned into a



■ **Figure 1** Left: A guardian at position p , storing all suffixes of $m_1 \dots m_{p-1}$ and all prefixes of $m_p \dots m_n$. Right: As soon as p escapes the range $[\frac{1}{4}n, \frac{3}{4}n]$ a new guardian (in green) is started at $\frac{n}{2}$.

1V-algorithm for $\text{prod}_{\mathcal{M}}$, if \mathcal{M} is a finite monoid, taking constant time on average and $\mathcal{O}(n)$ space. Tangwongsan, Hirzel, and Schneider presented a *worst-case* constant latency algorithm [28].

► **Theorem 2** (c.f. [28]). *Let \mathcal{M} be a fixed finite monoid (it is not part of the input). Then there is a 1V-algorithm for $\text{prod}_{\mathcal{M}}$ with word size $\mathcal{O}(\log n)$ and latency $\mathcal{O}(1)$.*

An immediate corollary of Theorem 2 is that every regular language L has a 1V-algorithm with word size $\mathcal{O}(\log n)$ and latency $\mathcal{O}(1)$. For this, one takes for the monoid \mathcal{M} in Theorem 2 the transformation monoid of a DFA for L . The transformation monoid of a DFA with state set Q and transition function $\delta : Q \times \Sigma \rightarrow Q$ is the submonoid of Q^Q (the set of all mappings on Q) generated by the functions $q \mapsto \delta(q, a)$, where $a \in \Sigma$. The monoid operation is the composition of functions: for $f, g \in Q^Q$ we define $fg \in Q^Q$ by $(fg)(q) = g(f(q))$ for all $q \in Q$.

In order to obtain for every regular language a 2V-algorithm with word size $\mathcal{O}(\log n)$, latency $\mathcal{O}(1)$, and space complexity $\mathcal{O}(n)$, we strengthen Theorem 2:

► **Theorem 3.** *Let \mathcal{M} be a fixed finite monoid. Then there is a 2V-algorithm for $\text{prod}_{\mathcal{M}}$ with word size $\mathcal{O}(\log n)$, latency $\mathcal{O}(1)$, and space complexity $\mathcal{O}(n)$.*

Proof sketch. We only consider the case, that the maximal window size n is known at advance (i.e., during initialization). The general case, where the maximal window size n is not known, can be dealt with Lemma 18 from Appendix A. We outline an algorithm with word size $\mathcal{O}(\log n)$, latency $\mathcal{O}(1)$, and space complexity $\mathcal{O}(n)$.

The limit n allows us to pre-allocate a circular bit array of size $\mathcal{O}(n)$, in which the window content $w = m_1 m_2 \dots m_\ell \in \mathcal{M}^*$ ($\ell \leq n$) is stored in a circular fashion and updated in time $\mathcal{O}(1)$. The current window length ℓ is stored in a register of bit length $\log n$. A constant number of pointers into w , including pointers to the first and to the last entry are also stored. Furthermore, to keep track of the product $m_1 m_2 \dots m_\ell$ under the allowed operations, some sub-products $m_i m_{i+1} \dots m_j$ for $i < j$ have to be stored as well. Storing all such products would require quadratic space. Instead, the algorithm stores (again in a circular fashion) for some index $p \in [1, \ell]$, called the *guardian*, all products $m_i m_{i+1} \dots m_p$ ($i \in [1, p-1]$) and $m_p m_{p+1} \dots m_j$ ($j \in [p+1, \ell]$); see Figure 1 for an illustration. If these products are only stored for all $i \in [k, p-1]$, $j \in [p+1, \ell]$ for some k, ℓ , then we speak of a *partial guardian*. As long as the guardian p satisfies $1 < p < n$, a push operations just adds one product and a pop operation removes one. A `leftpop()` decreases p and a `leftpush(a)` increases it. However, the algorithm only works correctly as long as the guardian is strictly between 1 and n .

Therefore, we enforce the invariant $p \in [\frac{1}{8}\ell, \frac{7}{8}\ell]$. To guarantee this invariant, we start a new guardian p' , initially set to $\frac{\ell}{2}$, whenever the old guardian p escapes the interval $[\frac{1}{4}\ell, \frac{3}{4}\ell]$. We need to make sure that the computation of the products for the new guardian is fast enough such that (i) p stays in $[\frac{1}{8}\ell', \frac{7}{8}\ell']$ while the guardian p' is partial, where ℓ' is the window size at that point, and (ii) p' stays in $[\frac{1}{4}\ell', \frac{3}{4}\ell']$. A simple calculation shows that it suffices if the guardian p' is complete after $\frac{1}{7}\ell$ steps where ℓ is the window size when p'

was initialized. Indeed, in worst case, the length of the string after $\frac{1}{7}\ell$ steps is $\frac{6}{7}\ell$. If the guardian is initially at position $\frac{1}{4}\ell$, it might afterwards be, again in worst case, at position $(\frac{1}{4} - \frac{1}{7})\ell = \frac{6}{56}\ell = \frac{1}{8} \times \frac{6}{7}\ell$. The dual case is analogous.

In each step (application of an operation), 8 new products are computed in a balanced² fashion, so that, after at most $\frac{\ell}{7}$ steps all products are available for the new guardian p' . In fact, if ℓ denotes the window size when the computation of the new guardian starts, after $\frac{\ell}{7}$ steps $\frac{8}{7}\ell$ products are computed, covering the potential window size after these steps.

The two guardians can be stored in registers of n size and for the two collections of partial products $\mathcal{O}(n)$ bits suffice. Moreover, all update operations can be carried out within a constant number of steps. ◀

Using again the transformation monoid of a regular language we obtain from Theorem 3:

► **Corollary 4.** *Every regular language L has a 2V-algorithm with word size $\mathcal{O}(\log n)$, latency $\mathcal{O}(1)$ and space complexity $\mathcal{O}(n)$.*

The lower bounds for the $\mathcal{O}(\log n)$ -time level can be summarized as follows; see [20] for the proof. Recall that non-uniform M -algorithms refer to the cell probe model from Section 2.

► **Theorem 5.** *For a regular language L and sliding-window model M , every non-uniform M -algorithm with word size 1 for L has latency at least $\log n - \mathcal{O}(1)$ for infinitely many n in each of the following three cases:*

- (a) $M \in \{1F, 1V\}$ and $L \notin \langle \text{Len}, \text{LI} \rangle$,
- (b) $M = 2V$ and $L \notin \text{Len} = \langle \text{Len} \rangle$,
- (c) $M = 2F$ and $L \notin \langle \text{Len}, \text{Fin} \rangle$.

In (b) and (c) the lower bound $\log n - \mathcal{O}(1)$ holds for all n .

► **Corollary 6.** *In each of the cases of Theorem 5, any M -algorithm with latency $\mathcal{O}(1)$ requires word size $\Omega(\log n)$.*

3.2 Sublogarithmic word size

There are two combinations of subclasses of regular languages and sliding window models, for which we obtain constant latency algorithms with word size $\mathcal{O}(\log \log n)$:

► **Theorem 7.** *Let L be a regular language and M a sliding-window model. If (i) $L \in \langle \text{Len}, \text{LI} \rangle$ and $M = 1V$ or (ii) $L \in \text{Len}$ and $M = 2V$, then there exists an M -algorithm for L with the following properties, where n is the maximum window size:*

- *The algorithm uses a RAM with a bit array of length $\mathcal{O}(\log n)$ and a constant number of pointers into the bit array. Each pointer can be stored in a register of length $\mathcal{O}(\log \log n)$.*
- *The latency of the algorithm is $\mathcal{O}(1)$.*

A detailed proof of Theorem 7 can be found in Appendix B. For both cases it is known that a sliding window algorithm (1V in case (i), 2V in case (ii)) with logarithmic space complexity exists [16]. These algorithms mainly use counters of maximal size n (the window size) and every window update makes a constant number of the following counter operations: increments, decrements, comparison of two counter values (where one of the two counters is always the same), and reset to zero. One could store the counters in the standard binary representation, but then the operations on counters would take time $\mathcal{O}(\log n)$. To overcome

² In principle, four products are computed for each side, but if the word grows towards one side, this can be reflected in the choice of the next products.

this problem, we use the counting techniques from [14], that allows to do the following counter operations in time $\mathcal{O}(\log \log n)$ for a fixed maximal counter size n : increment, decrement and comparison with a fixed number. We have to adapt this technique, since we also have to reset counters and compare counters. Moreover, since we work in the variable-size model, we have no limit on the size of the counters.

The lower bounds for the $\log \log n$ -time level follow again from known space lower bounds in the one-way model [16], see [20].

► **Theorem 8.** *For a regular language L and sliding-window model M , every non-uniform M -algorithm with word size 1 for L has latency at least $\log \log n - \mathcal{O}(1)$ for infinitely many n in each of the following two cases:*

(a) $M = 1V$ and L is not trivial,

(b) $M = 1F$ and $L \notin \langle \text{Len}, \text{SL} \rangle$,

In (a) the lower bound $\log n - \mathcal{O}(1)$ holds for all n .

► **Corollary 9.** *In each of the cases of Theorem 8, any M -algorithm with latency $\mathcal{O}(1)$ requires word size $\Omega(\log \log n)$.*

We finally turn to regular languages that have constant latency sliding window algorithms on a RAM with word size $\mathcal{O}(1)$. The proof of the following theorem can be found in [20].

► **Theorem 10.** *Let L be a regular language and M a model. In the following cases, L has a constant latency M -algorithm with word size $\mathcal{O}(1)$:*

(a) $M = 2V$ and L is trivial.

(b) $M = 2F$ and $L \in \langle \text{Len}, \text{Fin} \rangle$

(c) $M = 1F$ and $L \in \langle \text{Len}, \text{SL} \rangle$

4 Context-free languages

One natural question is which extensions of the regular languages admit constant latency sliding window algorithms. There is certainly no hope to go up to the whole class of context-free languages as this would yield a linear time algorithm for parsing context-free languages. We will show that, even for real-time deterministic context-free languages, there is also little hope to find a constant latency uniform fixed-size sliding window algorithm even though all deterministic context-free languages can be parsed in linear time. More precisely, we will construct a real-time deterministic context-free language for which we prove a lower bound of $\Omega(n^{1/2-o(1)})$ per update, under the OMV conjecture [23].

On the positive side, in this section we will present constant latency sliding window algorithms for the class of visibly pushdown languages and algorithms with logarithmic latency for deterministic 1-counter languages.

4.1 Lower bound for real-time deterministic context-free languages

First let us recall the online matrix-vector multiplication problem that we will use in our reduction. The *Online Matrix-Vector multiplication* (OMV) problem is the following: the input consists of a Boolean matrix $M \in \{0, 1\}^{n \times n}$ and n Boolean vectors $V_1, \dots, V_n \in \{0, 1\}^{n \times 1}$ and the vector $M \cdot V_i$ must be computed before the vector V_{i+1} is read. The OMV conjecture [23] states that a RAM with register length $\log n$ cannot solve the OMV problem in time $\mathcal{O}(n^{3-\epsilon})$ for any $\epsilon > 0$. The OMV conjecture implies tight lower bounds for a number of important problems, like subgraph connectivity, Pagh's problem, d -failure connectivity, decremental single-source shortest paths, and decremental transitive closure; see [23].

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \longrightarrow \$ m_{13} m_{12} m_{11} \$ m_{23} m_{22} m_{21} \$ m_{33} m_{32} m_{31} \# v_1 v_2 v_3$$

■ **Figure 2** Encoding of a matrix-vector product.

Recall that a real-time deterministic context-free language L is a language that is accepted by a deterministic pushdown automaton without ε -transitions. Hence, the automaton reads an input symbol in each computation step.

► **Lemma 11.** *There exists a real-time deterministic context-free language L such that any (uniform) 1F-algorithm for L with logarithmic word size and latency $t(n)$ yields an algorithm for the OMV problem with latency $\mathcal{O}(n^2) \cdot t(\mathcal{O}(n^2))$.*

Proof. Let n be the dimension of the OMV problem that we want to solve. We define the window size $m = 3n + n^2 + 1 = \Theta(n^2)$. The reduction will be based on a language $L \subseteq \{a, 0, 1, \$, \#\}^*$ that contains the word $a^j \text{enc}(M) \# \text{enc}(V) a^{n-j}$ (for $M \in \{0, 1\}^{n \times n}$, $V \in \{0, 1\}^{n \times 1}$, and $1 \leq j \leq n$) if and only if $M \cdot V$ contains a 1 on its j -th coordinate, for an encoding function enc that we now present. Since we do not care about strings that are not of this form, it does not matter which of them are in L . In fact, a string of the wrong form shall be in L , if and only if the automaton below accepts it.

The encoding of matrices and vectors is illustrated in Figure 2. A Boolean vector $V = [v_1, \dots, v_n]^T \in \{0, 1\}^{n \times 1}$ is encoded as the binary string $v_1 \cdots v_n \in \{0, 1\}^n$. A matrix $M \in \{0, 1\}^{n \times n}$ is encoded row by row, with the first row first, and the last row last. Each row starts with the dedicated symbol $\$$, followed by the encoding of the row (a word over the alphabet $\{0, 1\}$) in reverse order. Thus, the encoding of the j -th row starts with $M_{j,n}$ and ends with $M_{j,1}$.

We can construct a real-time deterministic pushdown automaton \mathcal{P} which accepts the word $a^j \text{enc}(M) \# \text{enc}(V) a^{n-j}$ for M, V, j as above, if and only if the j -th entry of the product $M \cdot V$ is 1: The pushdown automaton reads the a^j -prefix on the stack and skips to the j -th row encoding of the matrix by popping a from the stack on every read row delimiter $\$$. Then it reads the j -th row of M in reverse on the stack, skips to the encoding of V , and can verify whether the j -th row of M and V both have 1-bits at a common position.

Now let us suppose that we have a uniform fixed-size sliding window algorithm \mathcal{A} for L with logarithmic word size and latency $t(n)$. Given the matrix M , we initialize an instance of \mathcal{A} with window size $m = 3n + n^2 + 1 = \Theta(n^2)$ and fill the sliding window with $a^{2n} \text{enc}(M) \#$. This word has length $\mathcal{O}(n^2)$, so this initial preprocessing takes time $\mathcal{O}(n^2) \cdot t(\mathcal{O}(n^2))$.

From the data structure prepared with the window content $a^{2n} \text{enc}(M) \#$ we can get the last bit of the vector $M \cdot V_1$ by loading V_1 into the window with n updates each taking time $t(m) = t(\mathcal{O}(n^2))$. This yields the window content $a^n \text{enc}(M) \# \text{enc}(V_1)$, which is in L if and only if $(M \cdot V_1)_n = 1$. Then loading an a into the window we obtain the window content $a^{n-1} \text{enc}(M) \# \text{enc}(V_1) a$. It belongs to L if and only if $(M \cdot V_1)_{n-1} = 1$. We repeat the process to obtain all bits of $M \cdot V_1$. Computing $M \cdot V_1$ thus requires $2n - 1$ updates and thus time $\mathcal{O}(n) \cdot t(\mathcal{O}(n^2))$. After computing $M \cdot V_1$ we need to compute $M \cdot V_2$, and the other products $M \cdot V_i$ next, and for that we would like to reset the algorithm \mathcal{A} so that the data structure is prepared with the word $a^{2n} \text{enc}(M) \#$, again. Here the final “trick” is applied: instead of doing a new initialization for each vector, requiring $\mathcal{O}(n^2) \cdot t(\mathcal{O}(n^2))$ steps each time, we rather do a rollback: to this end, the sliding window algorithm \mathcal{A} is modified so that each time it changes the value of some register ℓ in memory, it writes ℓ and the old value of register

ℓ into a log. By rolling back this log it is able to undo all changes during the processing of V_1 . The extra running time for keeping the log and rolling back the computation is proportional to the number of changes in memory and thus needs time only $\mathcal{O}(n) \cdot t(\mathcal{O}(n^2))$.

Overall the time to deal with one vector is $\mathcal{O}(n) \cdot t(\mathcal{O}(n^2))$ which yields a total running time of $\mathcal{O}(n^2) \cdot t(\mathcal{O}(n^2))$ for OMV. \blacktriangleleft

Lemma 11 states that a sliding window for L with logarithmic word size and latency $t(n) = \mathcal{O}(n^{1/2-\epsilon})$ would yield an algorithm for OMV with a running time of $\mathcal{O}(n^{3-2\epsilon})$, which would contradict the OMV conjecture.

► **Corollary 12.** *There exists a fixed deterministic context-free language L such that, conditionally to the OMV conjecture, there is no (uniform) 1F-algorithm for L with logarithmic word size and latency $n^{1/2-\epsilon}$ for any $\epsilon > 0$.*

4.2 Visibly pushdown languages

In this section, we provide a constant latency 2V-algorithm for visibly pushdown languages. We first define visibly pushdown automata and their languages (for more details see [2]) and then show the upper bound result.

Visibly pushdown automata are like general pushdown automata, but the input alphabet is partitioned into call letters (that necessarily trigger a push operation on the stack), return letters (that necessarily trigger a pop operation on the stack), and internal letters (that do not change the stack). Formally, a *pushdown alphabet* is a triple $\Sigma = (\Sigma_c, \Sigma_r, \Sigma_{int})$ consisting of three pairwise disjoint alphabets: a set of *call letters* Σ_c , a set of *return letters* Σ_r and a set of *internal letters* Σ_{int} . We identify Σ with the union $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$. The set W of *well-nested* words over Σ is defined as the smallest set such that (i) $\{\varepsilon\} \cup \Sigma_{int} \subseteq W$, (ii) W is closed under concatenation and (iii) if $w \in W$, $a \in \Sigma_c$, $b \in \Sigma_r$ then also $awb \in W$. Every well-nested word over Σ can be uniquely written as a product of *Dyck primes* $D = \Sigma_{int} \cup \{awb : w \in W, a \in \Sigma_c, b \in \Sigma_r\}$ (W is a free submonoid of Σ^* that is freely generated by D). Note that every word $w \in \Sigma^*$ has a unique factorization $w = stu$ with $s \in (W\Sigma_r)^*$, $t \in W$ and $u \in (\Sigma_c W)^*$. To see this, note that the maximal well-matched factors in a word $w \in \Sigma^*$ do not overlap. If these maximal well-matched factors are removed from w then a word from $\Sigma_r^* \Sigma_c^*$ must remain (other one of the removed well-matched factors would be not maximal); see also [15, Section 5].

A *visibly pushdown automaton (VPA)* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \perp, q_0, \delta, F)$ where Q is a finite state set, Σ is a pushdown alphabet, Γ is the finite stack alphabet containing a special symbol \perp (representing the bottom of the stack), $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta = \delta_c \cup \delta_r \cup \delta_{int}$ is the transition function where $\delta_c : Q \times \Sigma_c \rightarrow (\Gamma \setminus \{\perp\}) \times Q$, $\delta_r : Q \times \Sigma_r \times \Gamma \rightarrow Q$ and $\delta_{int} : Q \times \Sigma_{int} \rightarrow Q$. The set of *configurations* Conf is the set of all words αq where $q \in Q$ is a state and $\alpha \in \perp(\Gamma \setminus \{\perp\})^*$ is the *stack content*. We define $\hat{\delta} : \text{Conf} \times \Sigma \rightarrow \text{Conf}$ as follows, where $p \in Q$ and $\alpha \in \perp(\Gamma \setminus \{\perp\})^*$:

- If $a \in \Sigma_c$ and $\delta(p, a) = (\gamma, q)$ then $\hat{\delta}(\alpha p, a) = \alpha \gamma q$.
- If $a \in \Sigma_{int}$ and $\delta(p, a) = q$ then $\hat{\delta}(\alpha p, a) = \alpha q$.
- If $a \in \Sigma_r$, $\gamma \in \Gamma \setminus \{\perp\}$, and $\delta(p, a, \gamma) = q$ then $\hat{\delta}(\alpha \gamma p, a) = \alpha q$.
- If $a \in \Sigma_r$ and $\delta(p, a, \perp) = q$ then $\hat{\delta}(\perp p, a) = \perp q$ (so \perp is not popped from the stack).

As usual we inductively extend $\hat{\delta}$ to a function $\hat{\delta}^* : \text{Conf} \times \Sigma^* \rightarrow \text{Conf}$ where $\hat{\delta}^*(c, \varepsilon) = c$ and $\hat{\delta}^*(c, wa) = \hat{\delta}(\hat{\delta}^*(c, w), a)$ for all $c \in \text{Conf}$, $w \in \Sigma^*$ and $a \in \Sigma$. In the following, we write $\hat{\delta}$ for $\hat{\delta}^*$. The *initial* configuration is $\perp q_0$ and a configuration c is *final* if $c \in \Gamma^* F$. A word $w \in \Sigma^*$ is *accepted* from a configuration c if $\hat{\delta}(c, w)$ is final. The VPA \mathcal{A} *accepts* w if w is

38:12 Low-Latency Sliding Window Algorithms for Formal Languages

accepted from the initial configuration. The set of all words accepted by \mathcal{A} is denoted by $L(\mathcal{A})$; the set of all words accepted from c is denoted by $L(c)$. A language L is a *visibly pushdown language (VPL)* if $L = L(\mathcal{A})$ for some VPA \mathcal{A} .

Now we are ready to state the main result of this section.

► **Theorem 13.** *Every visibly pushdown language L has a 2V-algorithm with latency $\mathcal{O}(1)$, space complexity $\mathcal{O}(n \log n)$ and word size $\mathcal{O}(\log n)$.*

For the proof of Theorem 13 we will use the 2V-algorithm for $\text{prod}_{\mathcal{M}}$ from the proof of Theorem 3 (\mathcal{M} is again a finite monoid). In the following, we call the data structure behind this algorithm a $\text{DABA}(\mathcal{M})$ data structure, where DABA stands for deamortized banker's algorithm (the name for the data structure in [28] used for the proof of Theorem 2). In the proof of Theorem 3, $\text{DABA}(\mathcal{M})$ stores a sequence of monoid elements $m_1, m_2, \dots, m_k \in \mathcal{M}$ and a $\text{prod}_{\mathcal{M}}$ -query returns the monoid product $m_1 m_2 \cdots m_k$. For our application of the $\text{DABA}(\mathcal{M})$ data structure to visibly pushdown languages in the next section, we have to store sequences of pointers p_1, p_2, \dots, p_k . Following pointer p_k we can determine a monoid element m_i (and some additional data values that will be specified below). When applying a $\text{prod}_{\mathcal{M}}$ -query to such a sequence of pointers, the monoid product $m_1 m_2 \cdots m_k \in \mathcal{M}$ is returned. In addition, we have the update-operations from 2V-model that allow to remove the first or last pointer or to add a new pointer at the beginning or end. All operations work in constant time.

Proof sketch of Theorem 13. Let us fix a (deterministic) VPA $\mathcal{A} = (Q, \Sigma, \Gamma, \perp, q_0, \delta, F)$ with $\Sigma = (\Sigma_c, \Sigma_r, \Sigma_{int})$. As usual, we denote with Q^Q the monoid of all mappings from Q to Q with composition of functions as the monoid operation (see also Section 3.1). Notice that \mathcal{A} can only see the top of the stack when reading return symbols. Therefore, the behavior of \mathcal{A} on a well-nested word is determined only by the current state and independent of the current stack content. We can therefore define a mapping $\phi: W \rightarrow Q^Q$ by $\hat{\delta}(\perp p, w) = \perp \phi(w)(p)$ for all $w \in W$ and $p \in Q$. Note that this implies $\hat{\delta}(\alpha p, w) = \alpha \phi(w)(p)$ for all $w \in W$ and $\alpha p \in \text{Conf}$. The mapping ϕ is a monoid morphism (recall that W is a monoid with respect to concatenation).

For the further consideration, it is useful to extend ϕ to a mapping $\phi: \Sigma^* \rightarrow Q^Q$, which will be no longer a monoid morphism. To do this we first define $\phi(a): Q \rightarrow Q$ for letters $a \in \Sigma_r \cup \Sigma_c$ by $\delta(p, a, \perp) = \phi(a)(p)$ for $a \in \Sigma_r$ and $\delta(p, a) = (\gamma, \phi(a)(p))$ for $a \in \Sigma_c$ (and some $\gamma \in \Gamma \setminus \{\perp\}$). Note that for a return letter a , $\phi(a)$ is the state transformation induced by the letter a on the stack only containing \perp . Consider now an arbitrary word $w \in \Sigma^*$. As mentioned above, there is a unique factorization

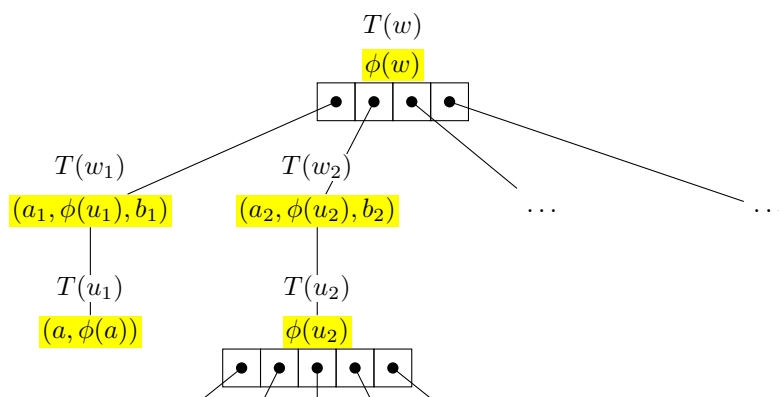
$$w = w_0 a_1 w_1 a_2 w_2 \cdots a_k w_k \in (W \Sigma_r)^* W (\Sigma_c W)^* \quad (1)$$

such that $k \geq 0$, $w_i \in W$ for all $0 \leq i \leq k$ and for some $s \in [0, k]$ (the *separation position*) we have $a_1, \dots, a_s \in \Sigma_r$ and $a_{s+1}, \dots, a_k \in \Sigma_c$. We then define the mapping $\phi(w): Q \rightarrow Q$ as $\phi(w) = \phi(w_0) \phi(a_1) \phi(w_1) \cdots \phi(a_k) \phi(w_k)$. Again, notice that the mapping $\phi: \Sigma^* \rightarrow Q^Q$ is not a monoid homomorphism. However, ϕ captures the behaviour of \mathcal{A} on a word w :

▷ **Claim 14.** For every word $w \in \Sigma^*$ and all states $q \in Q$ we have $\hat{\delta}(\perp q, w) = \alpha \phi(w)(q)$ for some stack content $\alpha \in \perp(\Gamma \setminus \{\perp\})^*$.

By this claim, shown in the long version [20], a variable-size sliding window algorithm for $L(\mathcal{A})$ only needs to maintain the state transformation $\phi(w)$ for the current window content w .

With a well-nested word $w \in W$ we associate a node-labelled ordered tree $T(w)$ as follows, where $\text{id}_Q \in Q^Q$ is the identity mapping on Q :



■ **Figure 3** The data structure of the 2V-algorithm for VPLs. A well-nested word w decomposes into Dyck primes $w = w_1 w_2 \cdots w_k$ where each w_i is either an internal letter or consists of a call letter a_i , a well-nested word u_i and a return letter b_i . The node of w stores the state transformation $\phi(w)$ together with a $\text{DABA}(Q^Q)$ instance, which maintains a list of the children w_i and their state transformations $\phi(w_i)$.

1. If $w = \varepsilon$ then $T(w)$ consists of a single node labelled with the pair $(\varepsilon, \text{id}_Q)$.
2. If $w = a \in \Sigma_{\text{int}}$ then $T(w)$ consists of a single node labelled with $(a, \phi(a))$.
3. If w is a Dyck prime aub with $a \in \Sigma_c$, $u \in W$ and $b \in \Sigma_r$, then $T(w)$ consists of a root node, labelled with $(a, \phi(w), b)$, with a single child which is the root of the tree $T(u)$.
4. If $w = w_1 \cdots w_k$ for Dyck primes w_1, \dots, w_k and $k \geq 2$, the root of $T(w)$ is labelled with $\phi(w)$ and it has the roots of the trees $T(w_1), \dots, T(w_k)$ k as children (from left to right). See Figure 3 for an illustration of the tree $T(w)$. We also speak of a tree $T(w)$ of type (i) (for $1 \leq i \leq 4$). Moreover, we say that a node v is of type (i) if the subtree rooted in v is of type (i) . Note that the type of node can be obtained from its label.

We have to implement several operations on such trees. In order to spend only constant time for each of the operations, we maintain the children v_1, \dots, v_k of a node v of type (4) as a $\text{DABA}(Q^Q)$ data structure which we denote by $\text{DABA}(v)$ (note that v_i is the root of the tree $T(w_i)$). This data structure is needed in order to maintain the value $\phi(w) = \phi(w_1) \cdots \phi(w_k)$ (the label of v). For the implementation of $\text{DABA}(v)$ we have to slightly extend the data structure from the proof of Theorem 3: there, every entry of the data structure stores an element of the monoid \mathcal{M} (here, Q^Q). Here, the entries of $\text{DABA}(v)$ are pointers to the children v_1, \dots, v_k . Note that from v_i we can obtain in constant time the monoid value $m_i := \phi(w_i) \in Q^Q$ as its label. The partial products $m_i \cdots m_j$ for the guardians as well as the additional $\mathcal{O}(1)$ many pointers to entries of the DABA data structure are treated as in the proof of Theorem 3. For the purpose of maintaining $\phi(w)$ only the monoid elements of m_1, \dots, m_k are relevant, but we use $\text{DABA}(v)$ also in order to store the list of children of v and hence the tree structure of $T(w)$. Intuitively, a tree $T(w)$ can be seen as a nested DABA data structure for the well-nested word w .

Assume now that the current window content is $w \in \Sigma^*$ and consider the unique factorization for w in (1) with the separation position $s \in [0, k]$. For a symbol $a \in \Sigma_c \cup \Sigma_r$ we write $\langle a \rangle$ for the pair $(\phi(a), a)$ below. Our 2V-algorithm stores on the top level two lists and a tree (here, again, every tree $T(w_i)$ is represented by a pointer to its root):

- the descending list $L_\downarrow = [T(w_0), \langle a_1 \rangle, T(w_1), \langle a_2 \rangle, \dots, T(w_{s-1}), \langle a_s \rangle]$
- the separating tree $T_s = T(w_s)$
- the ascending list $L_\uparrow = [\langle a_{s+1} \rangle, T(w_{s+1}), \dots, \langle a_k \rangle, T(w_k)]$.

These two lists are maintained by the DABA data structures DABA_\downarrow and DABA_\uparrow , respectively. These DABA data structures maintain the aggregated state transformations $\phi(w_0 a_1 w_1 a_2 \cdots w_{s-1} a_s)$ and $\phi(a_{s+1} w_{s+1} \cdots a_k w_k)$ from which, together with $\phi(w_s)$ (which can be obtained from the root of T_s) we can obtain the value $\phi(w)$, which, in turn, allows to check whether $w \in L(\mathcal{A})$ by Claim 14. Note that $L_\downarrow = []$ (the empty list) in case $s = 0$ and $L_\uparrow = []$ in case $k = s$. Since each of lists L_\downarrow and L_\uparrow can be empty, we will need all four types of operations (leftpop, rightpop, leftpush, and rightpush) for L_\downarrow and L_\uparrow . Therefore, the symmetric DABA data structure from Theorem 3 is really needed here (even if we only would aim for a 1V-algorithm).

Using this data structure, it is not difficult (but a bit tedious) to implement all window update operations in constant time, see [20]. Note that the above data structure uses $\mathcal{O}(n \log n)$ bits: we have to store $\mathcal{O}(n)$ many pointers of bit length $\mathcal{O}(\log n)$. This concludes our proof sketch of Theorem 13. \blacktriangleleft

4.3 Deterministic 1-counter automata

In this section we show that every deterministic 1-counter language has a 2V-algorithm with latency $\mathcal{O}(\log n)$ on a RAM with word size $\mathcal{O}(\log n)$. A deterministic 1-counter automaton (DOCA) \mathcal{A} is a deterministic finite automaton equipped with a single \mathbb{N} -counter. Its transition function δ specifies for every combination of current state q and emptiness condition of the counter ($= 0$ or > 0) either an ε -move (where no input symbol is read) or a unique a -move for every input symbol a . A move is specified by the next state q' and a counter update $d \in \{-1, 0, 1\}$ (where $d \geq 0$ if the counter is zero). To simplify the presentation in the following, we only work with DOCAs without ε -moves; see Appendix C for the general and formal definition. A *deterministic 1-counter language* is the language accepted by a DOCA.

The set of *configurations* of a DOCA \mathcal{A} with state set Q is $Q \times \mathbb{N}$. Every word $w \in \Sigma^*$ induces an *effect* $\hat{\delta}_w: Q \times \mathbb{N} \rightarrow Q \times \mathbb{N}$ where $\hat{\delta}_w(q, m) = (q', m')$ if and only if reading w starting from configuration (q, m) reaches configuration (q', m') . It is easy to see that $\hat{\delta}_w$ can be completely reconstructed from the restriction of $\hat{\delta}_w$ to the set $Q \times [0, |w|]$, since along every run of length $|w|$ starting with a counter of at least $|w|$, the counter can only become zero in the last step. Therefore, if $\hat{\delta}_w(q, |w|) = (q', m)$ then $\hat{\delta}_w(q, |w| + d) = (q', m + d)$. In the following the effect $\hat{\delta}_w$ of a word w is stored by its values on all configurations from $Q \times [0, |w|]$. For this, $\mathcal{O}(|w|)$ many registers of bit length $\mathcal{O}(\log |w|)$ suffice.

Observe that (i) given a configuration $c \in Q \times [0, n]$ and the effect $\hat{\delta}_w$ of a word of length at most n , one can compute $\hat{\delta}_w(c)$ in constant time, and (ii) given the effects $\hat{\delta}_u, \hat{\delta}_v$ of two words u, v of length at most n , one can compute the effect $\hat{\delta}_{uv}$ in time $\mathcal{O}(n)$. For DOCAs with ε -transitions we can also represent effects (slightly more involved) so that properties analogous to (i) and (ii) hold; see Appendix C for details.

► Theorem 15. *Every deterministic 1-counter language L has a 2V-algorithm with latency $\mathcal{O}(\log n)$, space complexity $\mathcal{O}(n \log^2 n)$ and word size $\mathcal{O}(\log n)$.*

Proof sketch. Let w be the current window and n its length and let \mathcal{A} be a deterministic 1-counter automaton for L . Effects of words are always taken with respect to \mathcal{A} . Our 2V-algorithm will preserve the following invariants:

1. The current window w is factorized into blocks B_0, \dots, B_m where B_i has length 2^{a_i} and for some $k \in [-1, m]$ we have $a_0 < a_1 < \dots < a_k$ and $a_{k+1} > a_{k+2} > \dots > a_m$. A block of length 2^a is also called a level- a block in the following.

2. Every level- a block B is recursively factorized into two level- $(a - 1)$ blocks that we call B 's left and right half. So the blocks B_0, \dots, B_m are the maximal blocks, i.e., every other block is contained in some B_i . The collection of all blocks is stored as a forest of full binary trees T_0, \dots, T_m , where T_i is the tree for block B_i .
3. For a certain time instant, let the age of a block B be the number of window updates that have occurred between the first point of time, where B is completely contained in the window and the current point of time. Note that B can either enter the window on the left or on the right end. If B is a level- a block and has age at least $2^a - 1$, then the effect of B must be completely computed and stored in the tree node corresponding to block B . We call such a block *completed*. In particular, the effect of a block of length 1 has to be available after one further update.

The following claim is an immediate consequence of the 3rd invariant.

▷ **Claim 16.** If a level- a block B has at least $2^a - 1$ many symbols to its left as well as at least $2^a - 1$ many symbols to its right in the window, then its age is at least $2^a - 1$, so it must be completed.

Thus, at every time instant, on every level i , there can be at most 2 non-completed blocks, namely the left most and right most block.

In Appendix D we show the following claim, from which it is easy to conclude that the query time is bounded by $\mathcal{O}(\log n)$.

▷ **Claim 17.** At each time instant the window w factors into $\mathcal{O}(\log n)$ completed blocks.

It remains to describe how to deal with window updates and how to preserve the invariants. We first consider the operation `leftpush(a)`. Let t be the current point of time. We measure time in terms of window updates; every window update increments time by 1. Let us assume that $a_k \geq a_{k+1}$ (if $a_k < a_{k+1}$ then one has to replace k by $k + 1$ below).

Basically we make an increment on the binary representation of the number $2^{a_0} + \dots + 2^{a_k}$. In other words, we consider the largest number $j \leq k$ such that $a_i = i$ for all $0 \leq i \leq j$ and replace the blocks B_0, \dots, B_j together with the new a in the window by a single level- $(j + 1)$ block B'_{j+1} . Thereby also one new level- i subblock B'_i of B'_j for every $0 \leq i \leq j$ arises: we have $B'_0 = a$ and $B'_i = B'_{i-1}B_{i-1}$ for $1 \leq i \leq j + 1$. By invariant 3, all blocks B_0, \dots, B_{j-1} are completed. If $j < k$ then B_j is also completed, but if $j = k$ (and $a_k > a_{k+1}$) then we can only guarantee that the left half of B_j is completed; its right half might be still non-completed. Let us assume that $j = k$, which is the more difficult case.

The effects of the new blocks B'_0, B'_1, \dots, B'_k can be computed bottom up as follows: The effect of $B'_0 = a$ is immediately computed when a arrives in the window. If the effect of B'_{i-1} is already computed, then using the equality $B'_i = B'_{i-1}B_{i-1}$ and using the fact that B_{i-1} is completed, we can compute the effect of B'_i in time $c \cdot 2^i$ for some constant c . In total, the computation of the effects of all blocks B'_0, B'_1, \dots, B'_k needs time $\sum_{0 \leq i \leq k} c \cdot 2^i \leq c \cdot 2^{k+1}$. We amortize this work over the next $2^k - 1$ window updates by doing a constant amount of work in each step. Thereby we ensure that at time instant $t + 2^i - 1$, the new blocks B'_0, \dots, B'_i are completed for every $0 \leq i \leq k$. In particular, at time instant $t + 2^k - 1$, the block B'_k is completed. But at that time instant, also B_k must be completed (if it is still in the window – due to pops B_k might have been disappeared in the meantime) and we can reach the goal of completing B'_{k+1} at time $t + 2^{k+1} - 1$ by still doing only a constant amount of work in each step. This ensures that invariant 3 is preserved for every new block B'_i . Moreover, before another level- i block arises at the left end of the window (which can only happen every 2^i steps), the new block B'_i is completed. Since the same arguments apply

to `rightpush(a)`, it follows that at every time instant, on each level i only two blocks are in the process of completion (one that was created on the left end and one that was created on the right end). Since there are at most $\log(n)$ levels and for the completion of each block a constant amount of work is done in each step, we get the time bound $\mathcal{O}(\log n)$.

For `leftpop()`, one has to remove all blocks along the leftmost path in the tree T_0 for block B_0 , which results in a sequence of smaller blocks of length $2^0, \dots, 2^{a-1}$ if B_0 is a level a -block. These blocks are already present in the tree T_0 . Some of the blocks on the leftmost path of T_0 might be not completed so far. Of course we stop the computation of their effects. Invariant 1 is preserved, also in case $k = -1$ (where $a_0 > a_1 > \dots > a_m$).

Since the data structure is symmetric, the operations `rightpop()` and `rightpush(a)` can be implemented in the same way. The algorithm uses space $\mathcal{O}(n \log^2 n)$: the dominating part are the values of the effect functions. On each level these are at most n numbers of bit length $\mathcal{O}(\log n)$. Moreover there are at most $\log n$ levels. This concludes the proof. ◀

5 Open problems

We conclude with some open problems: In Theorem 3 we assume that the size of the finite automaton for L is a constant. One should also investigate how the optimal word size and latency depend on the number of states of the automaton. For space complexity, this dependency is investigated in [16].

We showed that there is a real-time deterministic context-free language L such that, conditionally to the OMV conjecture, there is no 1F-algorithm for L with logarithmic word size and latency $n^{1/2-\epsilon}$ for any $\epsilon > 0$. The best known upper bound in this setting for deterministic context-free languages we are aware of is $\mathcal{O}(n/\log n)$. It is open, whether every deterministic context-free language has a one-way fixed-size sliding window algorithm with logarithmic word size and latency $n^{1-\epsilon}$ for some $\epsilon > 0$.

For every deterministic one-counter language L , we showed that there is a 2V-algorithm with latency $\mathcal{O}(\log n)$ and word size $\mathcal{O}(\log n)$. Here, it remains open, whether the latency can be further reduced, maybe even to a constant. Also, our space bound $\mathcal{O}(n \log^2 n)$ is not optimal. It would be nice to reduce it to $\mathcal{O}(n)$ without increasing the latency. The same problem appears for visibly pushdown languages, where our current space bound is $\mathcal{O}(n \log n)$ (with constant latency). Finally, it would be interesting to see, whether the 2V-algorithm with latency $\mathcal{O}(1)$ for visibly pushdown languages can be extended to the larger class of operator precedence languages [11, 8].

References

- 1 Charu C. Aggarwal. *Data Streams – Models and Algorithms*. Springer, 2007.
- 2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.
- 3 Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPICs*, pages 116:1–116:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.116.
- 4 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013. doi:10.1016/j.tcs.2012.12.028.
- 5 Ajesh Babu, Nutan Limaye, and Girish Varma. Streaming algorithms for some problems in log-space. In *Proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation, TAMC 2010*, volume 6108 of *Lecture Notes in Computer Science*, pages 94–104. Springer, 2010. doi:10.1007/978-3-642-13562-0_10.

- 6 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPICs*, pages 119:1–119:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.119.
- 7 Stanislav Böhm, Stefan Göller, and Petr Jancar. Equivalence of deterministic one-counter automata is NL-complete. In *Proceedings of the 45th ACM Symposium on Theory of Computing, STOC 2013*, pages 131–140. ACM, 2013. doi:10.1145/2488608.2488626.
- 8 Stefano Crespi-Reghizzi and Dino Mandrioli. Operator precedence and the visibly pushdown property. *Journal of Computer and System Sciences*, 78(6):1837–1867, 2012. doi:10.1016/j.jcss.2011.12.006.
- 9 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.
- 10 Eldar Fischer, Frédéric Magniez, and Tatiana Starikovskaya. Improved bounds for testing Dyck languages. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1529–1544. SIAM, 2018. doi:10.1137/1.9781611975031.100.
- 11 Robert W. Floyd. Syntactic analysis and operator precedence. *Journal of the ACM*, 10(3):316–333, 1963. doi:10.1145/321172.321179.
- 12 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPICs*, pages 43:1–43:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.43.
- 13 Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. Dynamic algorithms for the Dyck languages. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures, WADS 1995*, volume 955 of *Lecture Notes in Computer Science*, pages 98–108. Springer, 1995. doi:10.1007/3-540-60220-8_54.
- 14 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *Journal of the ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.
- 15 Moses Ganardi. Visibly pushdown languages over sliding windows. In Rolf Niedermeier and Christophe Paul, editors, *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPICs*, pages 29:1–29:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.29.
- 16 Moses Ganardi, Danny Hucce, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPICs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.STACS.2018.31.
- 17 Moses Ganardi, Danny Hucce, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPICs*, pages 18:1–18:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.FSTTCS.2016.18.
- 18 Moses Ganardi, Danny Hucce, and Markus Lohrey. Querying languages over sliding windows. *CoRR*, abs/1702.04376v1, 2017. arXiv:1702.04376.
- 19 Moses Ganardi, Danny Hucce, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPICs*, pages 127:1–127:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.127.
- 20 Moses Ganardi, Louis Jachiet, Markus Lohrey, and Thomas Schwentick. Low-latency sliding window algorithms for formal languages. *CoRR*, abs/2209.14835, 2022. arXiv:2209.14835.
- 21 Moses Ganardi, Artur Jez, and Markus Lohrey. Sliding windows over context-free languages. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPICs*, pages 15:1–15:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

- 22 Alejandro Grez, Filip Mazowiecki, Michał Pilipczuk, Gabriele Puppis, and Cristian Riveros. Dynamic Data Structures for Timed Automata Acceptance. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation, IPEC 2021*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2021.20.
- 23 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 24 Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, 1965. doi:10.1016/S0019-9958(65)90426-2.
- 25 Donald E. Knuth. *The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997. URL: <https://www.worldcat.org/oclc/312910844>.
- 26 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science, MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 412–423. Springer, 2011. doi:10.1007/978-3-642-22993-0_38.
- 27 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014. doi:10.1137/130926122.
- 28 Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. Low-latency sliding-window aggregation in worst-case constant time. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017*, pages 66–77. ACM, 2017. doi:10.1145/3093742.3095107.

A Additional material for the proof of Theorem 3

► **Lemma 18.** *Let T be a non-decreasing function and let A be a 2V-algorithm for a function f such that, when initialized with the empty window and a maximal window size of n ,*

- *A works on a unit-cost RAM with word size $\mathcal{O}(\log n)$,*
- *the initialization takes time $T(n)$,*
- *all later window operations run in time $T(n)$ and*
- *A stores in total at most $\mathcal{O}(n)$ bits.*

With such an A , we can build a 2V-algorithm A^ for f without the maximal window size limitation and with latency $\mathcal{O}(T(4n))$, space complexity $\mathcal{O}(n)$ and word size $\mathcal{O}(\log n)$.*

Proof. To get the statement of the lemma but with an amortized complexity bound, we could design the algorithm A^* to work just like any “dynamic array”: let us denote with A_n the version of the algorithm that works for window size up to n . Assume that we currently work with A_n . If the window size grows to n we switch to A_{2n} and if the window size shrinks to $n/4$ we switch to $A_{n/2}$. To do the switching, we transfer the current window content using *rightpush*-operations (we could also use *leftpush*) of A_{2n} or $A_{n/2}$ from the current version A_n to the new version. We call this a reset. If we reset to A_n we know that the window size was $n/2$. Therefore, the reset takes time $(n/2 + 1) \cdot T(n)$ (the $+1$ comes from the initialization which takes time $T(n)$). On the other hand, the next reset only happens if the window size grows to $2n$ or shrinks to $n/4$. Therefore, we make at least $n/4$ non-reset operations before the next reset happens. This leads to an amortized latency of $\mathcal{O}(T(4\ell))$, where ℓ is the current window size. The term $T(4\ell)$ comes from the fact that at each time instant we work with a version A_n , where n is at most a factor 4 larger than the actual window size.

To get non-amortized time bounds we need to maintain two instances of A at all time. One instance A_{cur} solving the problem (just like in the amortized case) and one instance A_{next} that we prepare in the background so that whenever we need to double or halve the maximal window size, the instance A_{cur} can just be replaced with A_{next} . Our algorithm thus stores two instances of A (A_{cur} and A_{next}), a copy of the current word w in the window (as an amortized circular buffer), some bookkeeping to know what is the size p of the prefix of w that has been loaded into A_{next} , the current size of w and the parameter n of the maximal window size of A_{cur} . Overall the algorithm stores $\mathcal{O}(\ell)$ bits where ℓ is the current window size ℓ . To prepare A_{next} , we decide that whenever the current window size is above $n/2$, we prepare A_{next} to work with $2n$ and whenever the window size is below $n/2$, we prepare A_{next} to work with $n/2$. Note that each time the window size crosses the $n/2$ threshold, our algorithm will reset A_{next} but each reset only takes time $T(2n)$ or $T(n/2)$, hence time $T(2n)$ at most (if T is monotone).

Propagating the effect of each window operation on A^* to A_{cur} and w is easy, we just apply the effect for w and call the right operation for A_{cur} . For A_{next} , if the window operation is a `leftpop()` or `leftpush(a)`, we carry out the same operation in A_{next} . This ensures that at any point during the algorithm, A_{next} stores a prefix of the current window whose length $p - 1$ (case `leftpop()`) or $p + 1$ (case `leftpush(a)`). If the operation is a `rightpop()` or `rightpush(a)`, we simply ignore it in A_{next} . Now, in both cases, i.e., for every update operation, we call `rightpush(a)` in A_{next} for up to 5 elements a ($w[p+1]$, $w[p+2]$, $w[p+3]$, $w[p+4]$, and $w[p+5]$) so that overall p (the length of the prefix of the window copied to A_{next}) increases by at least 4. Note that there might be less than 5 elements to push if A_{next} is almost ready.

Finally, note that because we switch between A_{cur} and A_{next} when the window size reaches $n/4$ or n and we restart the above copying process from A_{cur} to A_{next} whenever the window size becomes larger or smaller than $n/2$, we know that at least $n/4$ window updates must occur between the last restart and the actual switch from A_{cur} to A_{next} . Therefore by increasing the size of the prefix covered by A_{next} by 4 for each window update, we know that A_{next} covers the full window whenever we have to switch. Overall we do have a scheme with the right latency as we only do a constant number of calls to algorithm A for each window operation. ◀

B Proof of Theorem 7

We first consider the case that $L \in \text{Len}$ and $M = 2V$. In principle, it suffices to keep track of the current window size n and to check whether it is in some fixed semilinear set S . In fact, from L one compute a number N and two finite sets A and B such that a string is in L if and only if its length n satisfies (1) $n \geq N$ and $n \bmod N \in A$ or (2) $n < N$ and $n \in B$.

Obviously, the number n can be stored with $\mathcal{O}(\log n)$ bits. However, n needs to be incremented and decremented and compared, and the naive way of doing that may require the manipulation of $\log n$ bits for one operation, in worst case. Therefore we need to make use of a more sophisticated data structure that allows updates with a constant number of operations that manipulate only $\mathcal{O}(\log \log n)$ bits each.

Whenever $n \geq N$, the data structure uses a counter m that stores $n - N$, and a variable r of constant size that keeps track of $n \bmod N$ to check whether $n \bmod N \in A$. Whenever $n < N$, it stores n in a variable of constant size and maintains whether $n \in B$ holds.

During initialization, m is set to zero, and whenever $n < N$ it stays at zero. The challenging part is to maintain m if $n > N$ and to recognize whenever a phase with $n > N$ ends by reaching $m = 0$. This can be done using the counting techniques from [14]. In a

nutshell, the method basically works just as incrementing a binary number by, say, a Turing machine. However, to avoid long delays, it encodes the position of the head of the Turing machine in the string (by an underscore of the digit at that position) and each “move” of the Turing machine corresponds to an incremented number. E.g., the numbers 0,1,2,3,4 could be represented by $00\underline{0}$, $00\underline{1}$, $01\underline{0}$, $01\underline{1}$, $0\underline{10}$. It is not obvious how to determine the number represented by such a string. However, for our purposes it suffices to increment numbers, to compare them with a fixed constant number, and to initialize them with a fixed number.

The algorithm maintains the number m by a bit string s of length ℓ for some ℓ with $2^{\ell+1} - \ell - 2 \geq m$. The number ℓ is stored with $\mathcal{O}(\log \ell)$ bits. One position x in s is considered as marked and stored with $\mathcal{O}(\log \ell)$ bits as well. The representation of the number m by s and x is defined as follows (where the marked bit in s is underlined and u, v are bit strings):

- The number 0 is represented as $0^{\ell-1}\underline{0}$.
- If number t is represented by $u\underline{0}$ then $t + 1$ is represented by $u\underline{1}$.
- If number t is represented by $u0\underline{1}v$ then $t + 1$ is represented by $u10v$.
- If number t is represented by $u\underline{00}v$ then $t + 1$ is represented by $u0\underline{0}v$.
- If number t is represented by $u1\underline{1}v$ then $t + 1$ is represented by $u\underline{10}v$.
- If number t is represented by $\underline{1}0^{\ell-1}$ then $t + 1$ is represented by $1\underline{0}0^{\ell-1}$, and ℓ subsequently has to be incremented by 1.

The last of these cases does not appear in [14]; it is needed since in [14] m and ℓ are fixed, which is not the case in our application. It is a crucial observation that all bits to the right of the marked position are always zero. In particular this allows to identify when a situation of the form $\underline{1}0^{\ell-1}$ is obtained. We note that additional leading zeros do not spoil the representation of a number. Therefore ℓ needs never be decreased when the window size n and hence m is decreased.

One can show that in this way every number is represented in a unique way, ignoring unmarked leading zeros; see [14]. The above rules also specify how to increment and decrement m (for decrement one has to reverse the rules). Note that the rules modify s only in a local way; therefore they can be implemented in time $\mathcal{O}(1)$ on a RAM of word size $\mathcal{O}(\log \log n)$.

It remains to explain how to check whether $m = 0$ holds. To this end, an additional number is stored, which is the minimal position y in the bit string such that all positions to the left of it are 0 (the positions are numbered from right to left, i.e., the right-most position has number 0). For y , $\mathcal{O}(\log \ell)$ bits suffice, as well. Its manipulation is straightforward with the above rules.

If n is the maximal window size seen in the past, the algorithm stores a bit array of length $\mathcal{O}(\log n)$ for the bit string s and three registers of bit length $\mathcal{O}(\log \log n)$ for the numbers ℓ, x, y . This completes the description of the case that $L \in \text{Len}$ and $M = 2V$.

We now consider the case that $L \in (\text{Len}, \text{LI})$ and $M = 1V$. It suffices to consider the cases that $L \in \text{Len}$ and $L \in \text{LI}$: if we have a boolean combination of such languages, we can run the sliding window algorithms for these languages in parallel and combine their results according to the boolean formula. The case $L \in \text{Len}$ is covered by the first part of the proof (for $M = 2V$), so it suffices to consider the case $L \in \text{LI}$ and $M = 1V$. Hence, L is a left ideal. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a DFA for the reversal language $L^R = \{w^R : w \in L\}$ of the left ideal L where the reversal of a word $w = a_1a_2 \cdots a_n \in \Sigma^*$ is $w^R = a_n \cdots a_2a_1$. Since L is a left ideal we can assume that F contains a unique final state q_F , which is also a sink, i.e. $\delta(q_F, a) = q_F$ for all $a \in \Sigma$.

We use a simplified version of the $\mathcal{O}(\log n)$ -space *path summary algorithm* from [16]. A *path summary* is an unordered list $(Q_1, n_1)(Q_2, n_2) \cdots (Q_k, n_k)$ where the $Q_i \subseteq Q$ are pairwise disjoint and nonempty and the $n_i \in [0, n]$ are pairwise different. Note that $k \leq |Q|$

which is a constant in our setting. The meaning of a pair (Q_i, n_i) is the following, where w is the current window content: Q_i is the set of all states q for which n_i is the length of the shortest suffix s of w such that $\delta(q, s^R) = q_F$. Furthermore, if there exists no such suffix for a state q then the state q does not appear in any set Q_i . Clearly $w \in L$ if and only if $q_0 \in \bigcup_{i=1}^k Q_i$. Hence, it suffices to maintain a path summary for the current window content w . We first describe, how the operations can be handled, in principle.

For `leftpop()`, the algorithm removes the unique pair (Q_i, n_i) with $n = n_i$ if it exists. For `rightpush(a)`, it replaces each pair (Q_i, n_i) by $(\{p \in Q \setminus \{q_F\} : \delta(p, a) \in Q_i\}, n_i + 1)$ if $\{p \in Q \setminus \{q_F\} : \delta(p, a) \in Q_i\}$ is non-empty, otherwise the pair is removed from the path summary. Finally, the pair $(\{q_F\}, 0)$ is added to the path summary.

It is easy to verify that the path summary is correctly maintained, in this way. However, some care is needed to guarantee the bounds claimed in the statement of the theorem.

We first observe that the length k of the path summary is bounded by the constant $|Q|$. Thus the algorithm stores at most $|Q|$ many pairs (Q_i, n_i) . They can be stored in a bit array of length $\mathcal{O}(\log n)$ that is divided into $|Q|$ many chunks. Every chunk stores one pair (Q_i, n_i) . The state set $Q_i \subseteq Q$ is stored with $|Q|$ many bits. The number n_i can be stored with $\log n$ many bits. Every chunk has an additional *activity bit* that signals whether the chunk is active or not. This is needed since in the path summary the algorithm has to be able to remove and add pairs (Q_i, n_i) . If the activity bit is set to 0 then the chunk is released and can be used for a new pair, later on. In addition the algorithm stores the window size n .

We have seen in the case $L \in \text{Len}$, how the kind of counters that are needed for a path summary and the window size can, in principle, be implemented such that a single update works with a constant number of operations that manipulate only $\mathcal{O}(\log \log n)$ bits. However, there are still two challenges that need to be mastered: (1) the counters n_i need to be compared with the number n (the window size), which itself can change, and (2) when a chunk is deactivated, its counter n_i may represent any number, but when its re-activated it should be 0, again. Towards (1), the algorithm maintains a second counter, m_i for each chunk, which is supposed to represent $n - n_i$. Thus, to test $n = n_i$ it suffices to check whether $m_i = 0$ and we know from the `Len`-case how this can be done. Note that for an `leftpop()`, m_i must be decremented and for a `rightpush(a)`, m_i does not change.

For (2), the algorithm uses a “lazy copying” technique (from 0 and from n , respectively). The algorithm stores two additional numbers b_i and d_i of size $\mathcal{O}(\log \log n)$ for each counter m_i and one additional number a_i of size $\mathcal{O}(\log \log n)$ for each counter n_i .

We first describe, how to deal with n_i . If chunk i becomes activated, n_i is supposed to be initialized to 0, but the actual memory that it occupies might consist of arbitrary bits (inherited from the previous counter for which the chunk was used). Overwriting these bits would require $\Theta(\log n)$ bit operations, but the algorithm only can manipulate $\Theta(\log \log n)$ bits per operation. Therefore, n_i is represented by some bits of chunk i and all other bits are considered as being zero (independently of what they actually are). The additional number a_i tells how many of the last bits of the chunk for n_i are valid for the representation of n_i . When chunk i is activated, n_i should be 0 and therefore a_i can be set to 0, signifying that all bits of n_i are zero. In each subsequent step, a_i is incremented by 2 and two additional positions in the chunk are set to zero until a_i equals the length of the bit string.

For m_i , we use a similar technique but the situation is slightly more complicated since m_i should be initially set to n . If the marked position of n is position k then both numbers b_i and d_i are set to k . This indicates that all positions of m_i up to b_i are as in n and all positions from d_i on are as in n . Which clearly means that m_i is n , as required. Subsequently, b_i is decremented by 2 in each step, d_i is incremented by 2 in each step and the respective bits are copied from n to the chunk. Note that these copied bits of n have not changed their values since chunk i has been activated (this holds since we copy 2 bits in each step).

C Additional details on deterministic 1-counter automata

To extend the algorithm from Section 4.3 to DOCAs with ε -transitions we work with a slightly different, yet equivalent, definition of DOCAs from [7]: A *deterministic 1-counter automaton* is a tuple $\mathcal{A} = (Q_s, Q_r, \Sigma, \delta, \pi, \rho, q_0, F)$, where Q_s is a finite set of stable states, Q_r is a finite set of reset states ($Q_s \cap Q_r = \emptyset$), Σ is a finite input alphabet, $\delta : Q_s \times \Sigma \times \{0, 1\} \rightarrow (Q_s \cup Q_r) \times \{-1, 0, 1\}$ is the transition function, $\pi : Q_r \rightarrow \mathbb{N}$ maps every reset state q to a period $\pi(q) > 0$, $\rho : \{(q, k) \mid q \in Q_r, 0 \leq k < \pi(q)\} \rightarrow Q_s$ is the reset mapping, q_0 is the initial state, and $F \subseteq Q$ is the set of final states. It is required that if $\delta(q, a, i) = (q', j)$ then $i + j \geq 0$ to prevent the counter from becoming negative. Let $Q = Q_s \cup Q_r$. The set of configurations of \mathcal{A} is $Q \times \mathbb{N}$. For a configuration (q, m) and $k \in \mathbb{N}$ we define $(q, m) + k = (q, m + k)$. Intuitively, \mathcal{A} reads an input letter whenever it is in a stable state and changes its configuration according to δ . If \mathcal{A} is in a reset state q it resets the counter to zero and goes into a stable state that is determined (via the reset mapping ρ) by $m \bmod \pi(q)$ when m is the current counter value. Formally, we define the mappings $\hat{\delta} : Q \times \mathbb{N} \times \Sigma \rightarrow Q \times \mathbb{N}$ and $\hat{\rho} : Q \times \mathbb{N} \rightarrow Q_s \times \mathbb{N}$ as follows, where $\text{sign} : \mathbb{N} \rightarrow \{0, 1\}$ is the signum function restricted to the natural numbers:

- If $q \in Q_r$ and $m \in \mathbb{N}$ then $\hat{\rho}(q, m) = (\rho(q, m \bmod \pi(q)), 0)$.
- If $q \in Q_s$ and $m \in \mathbb{N}$ then $\hat{\rho}(q, m) = (q, m)$.
- If $q \in Q_s$ and $m \in \mathbb{N}$ then $\hat{\delta}(q, a, \text{sign}(m)) = \hat{\rho}(\delta(q, a, \text{sign}(m)) + m)$.
- If $q \in Q_r$ and $m \in \mathbb{N}$ then $\hat{\delta}(q, m) = \hat{\delta}(\hat{\rho}(q, m))$ (note that $\hat{\rho}(q, m) \in Q_s \times \{0\}$, for which $\hat{\delta}$ has been defined in the previous point).

We extend $\hat{\delta}$ to a function $\hat{\delta} : Q \times \mathbb{N} \times \Sigma^* \rightarrow Q \times \mathbb{N}$ in the usual way: $\hat{\delta}(q, x, \varepsilon) = (q, x)$ and $\hat{\delta}(q, x, aw) = \hat{\delta}(\hat{\delta}(q, x, a), w)$. Then, $L(\mathcal{A}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, 0, w) \in F \times \mathbb{N}\}$ is the language accepted by \mathcal{A} . A language L is a deterministic 1-counter language if $L = L(\mathcal{A})$ for some deterministic 1-counter automaton \mathcal{A} .

Using the function $\hat{\delta}$ we can define also runs of \mathcal{A} (we speak of \mathcal{A} -runs) on a word w in the usual way: For a configuration (q, m) and a word $w = a_1 a_2 \dots a_k$ the unique \mathcal{A} -run on the word w starting in (q, m) is the sequence of configurations $(q_0, m_0), (q_1, m_1), \dots, (q_k, m_k)$ where $(q_i, m_i) = \hat{\delta}(q_0, 0, a_1 \dots a_i)$. We denote this run with $\text{run}(q, m, w)$.

For a word $w \in \Sigma^*$ we define the *effect* $\hat{\delta}_w : Q \times \mathbb{N} \rightarrow Q \times \mathbb{N}$ by $\hat{\delta}_w(q, m) = \hat{\delta}(q, m, w)$. It specifies how w transforms configurations. It turns out that $\hat{\delta}_w$ can be completely reconstructed from restriction $\hat{\delta}_w \upharpoonright_{Q \times [0, |w| + p]}$ of $\hat{\delta}_w$ to the set $Q \times [0, |w| + p]$, where p is the least common multiple of all $\pi(q)$ for $q \in Q_r$. To see this, assume that (q, m) is a configuration with $m > |w| + p$. If in $\text{run}(q, |w| + p, w)$ no state from Q_r is visited and $\hat{\delta}_w(q, |w| + p) = (q', m')$, then we have $\hat{\delta}_w(q, m) = (q', m' + m - |w| - p)$: basically, we obtain $\text{run}(q, m, w)$ by shifting $\text{run}(q, |w| + p, w)$ upwards by $m - |w| - p$. On the other hand, if a reset state that appears in $\text{run}(q, |w| + p, w)$ then $\hat{\delta}_w(q, m) = \hat{\delta}_w(q, |w| + i)$, where i is any number in $[0, p]$ such that $|w| + i \equiv m \pmod{p}$.

In the following, we always assume that the effect $\hat{\delta}_w$ of a word w is stored by $\hat{\delta}_w \upharpoonright_{Q \times [0, |w| + p]}$, for which $\mathcal{O}(|w|)$ many registers of bit length $\mathcal{O}(\log |w|)$ suffice. Given the effects $\hat{\delta}_u$ and $\hat{\delta}_v$ of two words u, v of length at most n (stored in $\mathcal{O}(n)$ many registers of bit length $\mathcal{O}(\log n)$), we can compute the effect $\hat{\delta}_{uv}$ in time $\mathcal{O}(n)$ on a RAM with word size $\mathcal{O}(\log n)$. The computation of $\hat{\delta}_{uv}$ on an argument from $Q \times [0, |uv| + p]$ only involves simple arithmetic operations and can be done in constant time.

D Proof of Claim 17

We show the claim for the case $a_k > a_{k+1}$ (the cases $a_k < a_{k+1}$ and $a_k = a_{k+1}$ can be treated analogously). The middle block B_k can be factorized into $2a_k$ completed blocks, since if we consider the binary tree T_k for B_k , then only the blocks on the left most and right most root-

leaf path of T_k can be non-completed. This follows from Claim 16 since each level- b subblock of B_k that does not belong to the left-most or right-most path in T_k has another level- b subblock to its left as well as to its right. For the blocks B_{k+1}, \dots, B_m we show that $B_{k+1} \dots B_m$ can be factorized into at most $a_{k+1} + m - k - 1 \in \mathcal{O}(\log n)$ completed blocks, whereas $B_0 \dots B_{k-1}$ can be factorized into at most $a_{k-1} + k - 1 \in \mathcal{O}(\log n)$ completed blocks. Consider the level- a_i block B_i for some $1 \leq i \leq k - 1$. Every level- b subblock of B_i that does not belong to the left most path in T_i has at least $2^b - 1$ many symbols to its right (namely the block B_k) as well as to its left (namely another level- b subblock of B_i), and is therefore completed. But for blocks on the left most path in T_i the same is true when they belong to some level $b \leq a_{i-1}$, because then the block B_{i-1} is to its left. Hence, there are at most $a_i - a_{i-1}$ non-completed blocks in T_i and they form an initial part of the left most path. Removing those blocks from T_i leads to a factorization of B_i into at most $a_i - a_{i-1} + 1$ completed blocks. Finally, for the first block B_0 we obtain with the same argument a factorization into at most a_0 completed blocks. By summing over all block B_i ($0 \leq i \leq k - 1$) we obtain a factorization of $B_0 \dots B_{k-1}$ into at most $a_0 + \sum_{1 \leq i \leq k-1} (a_i - a_{i-1} + 1) = a_{k-1} + k - 1$ completed blocks. For $B_{k+1} \dots B_m$ we can argue analogously. The above arguments also show how to compute a factorization of the window into completed blocks. This factorization can be represented by a sequence of pointers to the tree nodes that correspond to the completed blocks in the factorization.

The Design and Regulation of Exchanges: A Formal Approach

Mohit Garg ✉

University of Bremen, Bremen, Germany
University of Hamburg, Hamburg, Germany

Suneel Sarswat ✉

Tata Institute of Fundamental Research, Mumbai, India

Abstract

We use formal methods to specify, design, and monitor continuous double auctions, which are widely used to match buyers and sellers at exchanges of foreign currencies, stocks, and commodities. We identify three natural properties of such auctions and formally prove that these properties completely determine the input-output relationship. We then formally verify that a natural algorithm satisfies these properties. All definitions, theorems, and proofs are formalized in an interactive theorem prover. We extract a verified program of our algorithm to build an automated checker that is guaranteed to detect errors in the trade logs of exchanges if they generate transactions that violate any of the natural properties.

2012 ACM Subject Classification Applied computing → Online auctions; Software and its engineering → Software verification and validation; Software and its engineering → Correctness

Keywords and phrases Double Auctions, Formal Specification and Verification, Financial Markets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.39

Supplementary Material *Other (Coq-Formalization)*: <https://github.com/suneel-sarswat/cda>

1 Introduction

Continuous double auctions are widely used to match buyers and sellers at market institutions such as foreign exchange markets, cryptocurrency exchanges, stock exchanges, and commodities exchanges. Increasingly, the exchanges deploy automated computer systems for conducting trades, which often receive a huge number of trade requests, especially in presence of high-frequency algorithmic traders. A minor bug in the computer system of a major exchange can have a catastrophic effect on the overall economy. To ensure that the exchanges work in a fair and orderly manner, they are subject to various regulatory guidelines. There have been a number of instances where the exchanges have been found violating regulatory guidelines [18, 17, 14, 16]. For example, it was observed that NYSE Arca failed to execute certain trades in violation of the stated rules [18], which led to the U.S. Securities and Exchange Commission imposing a heavy penalty on the New York Stock Exchange.

The above example is an instance of a program not meeting its specification, where the exchange’s order matching algorithm is the program and the exchange rules and regulatory guidelines form the broad specifications for the program. The reasons for such violations are generally twofold. Firstly, the exchange rules and the regulatory guidelines are not presented in a formal language; thus, they tend to be rather vague and ambiguous. Even if they are stated unambiguously, it is not clear whether they are consistent; they lack a formal proof of consistency. Secondly, the fidelity of the program implementing the specifications is traditionally based on repeatedly testing the software on large data sets and removing all bugs that get detected. Although this approach irons out many bugs, it is not foolproof; in particular, many bugs that surface only in rare scenarios may easily remain hidden in the



© Mohit Garg and Suneel Sarswat;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 39; pp. 39:1–39:21



Leibniz International Proceedings in Informatics

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

program. A related concern that is often missed is that a faulty program might continue to make mistakes without anyone noticing or even realizing that they are mistakes. This is especially relevant in the context of exchanges, where individual traders have a limited view of the system and may not realize that the exchange is making mistakes, and the regulators might overlook them, probably because they are not the kinds of mistakes that they look for or are brought to their notice.

In this work, we introduce a new framework for exchange design and regulation that addresses the above concerns. We elucidate our approach for an exchange that implements the widely used continuous double auction mechanism to match buy and sell requests. To this end, we develop the necessary theoretical results for continuous double auctions which then allows us to apply the formal methods technology, which has classically been applied to safety-critical systems like nuclear power plants [21], railway signaling systems, flight control software, and in hardware design (see [19] for a survey).

More specifically, we consider a general model for continuous double auctions and identify three natural properties, namely **price-time priority**, **positive bid-ask spread**, and **conservation**, that are necessary for any online algorithm implementing continuous double auction to possess. We then show that any algorithm satisfying these three properties for each input must have a “unique” output at every time step, implying that these three properties are sufficient for specifying continuous double auctions, and there is no need to impose any further requirements on an algorithm implementing such an auction. We then develop an algorithm and formally prove that it satisfies these three properties. This establishes the consistency of the three properties.

All our proofs are machine-checked; all the notions, definitions, theorems, and proofs are formalized in the popular Coq proof assistant [15]. Finally, using Coq’s program extraction feature, we obtain an OCaml program of our verified algorithm for implementing continuous double auctions. Such a verified program is guaranteed to be bug-free and can be directly used at an exchange. Additionally, enabled by the uniqueness theorem we prove, a verified program can be used to check for errors automatically in an existing exchange program by comparing the outputs of the verified program against that of the exchange program. Such a verified checker goes over the trade logs of the exchange and is guaranteed to detect a violation if the exchange output violates any of the three properties, and this alleviates the final concern we mentioned above. This can be a crucial tool for regulators. As an application, using our verified program we check for errors in real data collected from an exchange.

Thus, our approach can be summed up as follows: The specification for an exchange should consist of a set of provably consistent properties, which are preferably few and simple. Furthermore, these properties should be rich enough so that one can build an automated checker that can always detect a violation of the specification by going over the logs of the exchange if one exists.

We begin by briefly describing the trading process using continuous double auctions.

1.1 Overview of continuous double auctions

A double auction is a process to match multiple buy and sell orders for a given product at a marketplace. Potential buyers and sellers of a product place their orders at the market institution for trade. Buy and sell orders are referred to as bids and asks, respectively. The market institution on receiving orders is supposed to produce transactions. Each transaction is between a bid and an ask and consists of a transaction price and a transaction quantity.

More specifically, each order consists of an order id, timestamp, limit price, and maximum quantity. Order ids are used to distinguish orders (and hence are unique for each order). Timestamps represent the arrival times of the orders. For a fixed product, the timestamps of the orders are distinct.¹ The limit price of an order ω is the maximum (if ω is a bid) or minimum (if ω is an ask) possible transaction price for each transaction involving ω . The maximum quantity q of an order ω is the number of units of the product offered for trade, i.e., the sum of the transaction quantities, where ω participates in, is at most q .

Double auctions are used to generate transactions systematically. There are two common types of double auctions that are used at various exchanges: call auctions [20, 9, 22] and continuous double auctions. We discuss call auctions in Section 1.3. In a system implementing continuous double auctions, buyers and sellers may place their orders at any point in time. On receiving an order ω , the system instantly tries to match ω with fully or partially unmatched orders that arrived earlier (“resident orders”) and output transactions, before it proceeds to process any other incoming orders. If there are multiple orders with which ω can be matched, then the system prioritizes those matchable orders based on **price-time priority** (first the orders are ranked by the competitiveness of their price; orders with the same price are then ranked by their arrival time, i.e., as per their timestamps). If the total transaction quantities of the transactions generated by ω exhaust the maximum quantity of ω , then it leaves the system completely. Else ω with its remaining unmatched quantity is retained by the system for potential future matches; we refer to such unmatched orders as resident orders. A resident order leaves the system when its quantity gets fully exhausted.

The general model we work with has three primitive instruction types: buy, sell, and delete.² That is, apart from buy and sell instructions, it is possible to delete a resident order by giving a “delete id” instruction to the system. The system maintains two main logbooks: an order book and a trade book. The order book is a list of all buy, sell, and delete instructions that the system received, ordered by their timestamps. The trade book consists of the list of all transactions generated by the system in the order in which they were generated.

We now describe three natural properties of a system implementing continuous double auctions that are relevant to our formalization.

- **Positive bid-ask spread.** This property represents the “inertness” of resident orders; at any point in time, the resident orders are not matchable to each other, i.e., the limit price of each resident bid is strictly less than the limit price of each resident ask. This follows immediately from the fact that an order becomes resident only when it cannot be matched any further with the existing resident orders (or when there are no matchable resident orders at all).
- **Price-time priority.** When a new order arrives, the system tries to match it with the resident orders and generate transactions. If a less-competitive resident order participates in one of these transactions, then all current resident orders which are more competitive must get fully traded in these transactions, where competitiveness is decided based on **price-time priority**. This is a direct consequence of the matching process described above.
- **Conservation.** This property expresses the “honesty” of the system. Roughly speaking, it states that the system should not lose, create, or modify orders arbitrarily. For example, when a new incoming order gets partially traded with existing resident orders,

¹ In real systems, even if multiple orders arrive at the same time, they are entered into the system one by one. How this is exactly achieved is beyond the scope of this work.

² In Section 8 we describe how certain other instruction types can be converted to these primitives.

it becomes resident with its maximum quantity precisely being the difference between its original maximum quantity and the total quantity of the transactions that are generated. Furthermore, the timestamp, id, and limit price of the resident order must remain unchanged.

One might feel that many properties would be necessary for specifying “conservation” formally. On the contrary, later we will see that the above three properties can be stated rather succinctly once we set up our definitions appropriately. Also, we will show that any algorithm that implements these three properties, on any input (order book), will produce a “unique” output at every point in time, implying that these properties completely characterize continuous double auctions. One can learn more about double auctions from [5, 12, 3, 4].

1.2 Results

After setting up the definitions appropriately, we formally represent the three natural properties of continuous double auctions that we discussed above, namely **price-time priority**, **positive bid-ask spread**, and **conservation**. We are then able to prove the following results.

- **Maximum matching.** We first show in Lemma 4 that any algorithm that satisfies **positive bid-ask spread** and **conservation**, for each incoming order will generate a set of transactions (“matching”) whose total quantity will be maximum, i.e., it will be at least the total quantity of any other feasible set of transactions.
- **Local Uniqueness.** In Theorem 5 we show that any two processes that satisfy the aforementioned three properties, for any set of resident orders and an incoming instruction that satisfy certain technical conditions, will essentially generate the same matching, and the resident orders that remain at the end of processing this instruction will also be the same. To prove this theorem we make use of Lemma 4.
- **Global Uniqueness.** We then show in Theorem 6 that any two processes that satisfy the three properties, given any order book as input, at any point in time will essentially generate the same matching, and the resident orders that remain will also be the same. We prove this theorem by lifting Theorem 5 via an induction argument.
- **Correctness of `Process_instruction`.** We then design an algorithm for continuous double auctions which we refer to as `Process_instruction` and then in Theorem 7 show that it satisfies the three properties.

To use the above results in a practical setting, we extract an OCaml program of our verified algorithm `Process_instruction`, using Coq’s code extraction feature, and then use it to check for errors in real data from an exchange by running it on order books and then comparing the outputs with the corresponding trade books. Our global uniqueness theorem implies that if the exchange program has the three natural properties, then the output of our program should “match” with the output of the exchange. If they do not match, then at least one of the three properties must be violated by the exchange program.

Our Coq formalization uses about 6000 lines of new code, which includes about 250 lemmas and 80 definitions and functions, and is available in the accompanying supplementary materials [1].

1.3 Related work

Mavroudis and Melton [7] study fairness, transparency, and manipulation in exchange systems. They argue that idealized market models do not capture infrastructural inefficiencies and their simple mechanisms are not robust enough to withstand sophisticated technical manipulation

attacks; as a result, the theoretical fairness guarantees provided by such exchanges are not retained in reality. To address this problem, they propose the LIBRA system that achieves a relaxed notion of fairness by making clever use of randomness in routing orders to the matching engine. Our work focuses on a complementary and an arguably more basic aspect of exchanges; once the system assigns unique timestamps, even if they are “inaccurate” or “unfair” due to infrastructural inaccuracies or other reasons, with respect to those timestamps the matching engine should respect the desired properties; our work focuses on specifying and checking the correctness of the matching engine.

There have been some important works on formalizing financial systems, double auctions, and theorems in economics. We briefly mention a few of them. In an influential work [11], Passmore and Ignatovich highlight the need for formal verification of financial algorithms and suggest various open problems in the field. In response to these challenges, they designed Imandra [10], a specialized formal verification system and programming language designed to reason about properties of algorithms that may be proved, refuted, or described.

Cervesato, Khan, Reis, and Žunić [2] present a declarative and modular specification for an automated trading system for continuous double auctions in a concurrent linear framework (CLF) and implemented it in a CLF type checker that also supports executing of CLF specifications. Among other things, they were able to establish that their system is never in a locked or crossed state, which is equivalent to the **positive bid-ask spread** property mentioned above.

Wurman, Walsh, and Wellman [20] deal with the theory and implementation of flexible double auctions for electronic exchanges. In particular, the authors analyze the incentive compatibility of such auctions. Kaliszky and Parsert [6] introduce a formal micro-economic framework and formally prove the first welfare theorem leading to a more refined understanding of the theorem than was available in the economics literature.

Comparison with call auctions. Our work is closely related to the work of Raja, Singh, and Sarswat [8], which formalizes call auctions. As mentioned earlier, call auctions form a class of double auctions, which are “non-continuous” or “one-shot” in nature. In call auctions, orders are collected from buyers and sellers for a fixed duration of time, at the end of which the orders are matched simultaneously to generate transactions. In [8] call auctions are formalized and certain uniqueness theorems are obtained. Unlike call auctions, prior to our work, for continuous double auctions the specifications were not concisely stated in the literature, hindering formal development as also noted in [8].

It is interesting to compare call auctions with one time instant of continuous double auctions when a new order arrives in presence of resident orders. In call auctions two different objectives are usually considered: 1. Maximum matching: produce transactions that maximize the trade volume (the sum of transaction quantities); 2. Maximum uniform-price matching: produce transactions that maximize the trade volume subject to the constraint that all the transaction prices are the same. It turns out, there are instances where meeting these objectives leads to different matchings and trade volumes. In either case, after the transactions are produced, the orders or parts of the order that remain unmatched are not-matchable, i.e., they have **positive bid-ask spread**. Compare this with one time instant of continuous double auction. As we show in this work, **positive bid-ask spread** already makes the process unique and can be seen as a special case of a call auction where the output matching meets both the objectives (1 and 2) simultaneously.³ In the context of continuous

³ For this, we need a slightly relaxed definition of uniformity: we say a matching has a uniform price if there exists a common price that can be set for all transactions in the matching such that no limit price of the participating orders is breached.

double auctions, this leads to an arguably much simpler requirement of **positive bid-ask spread** as opposed to the requirements of maximum matching or maximum uniform-price matching. Additionally, we derive a stronger uniqueness theorem for continuous double auctions: for each bid-ask pair, the total transaction quantity between them is unique; in contrast, for call auctions, for each order, the total transaction quantity involving that order is unique. Where continuous double auctions completely differ from call auctions is in their online setting, which adds a new layer of complexity to the formalization.

1.4 Organization of the rest of the paper

For understanding the rest of the paper, we do not assume that the reader has any prior knowledge or expertise in formalization or using interactive theorem provers like Coq. For an interested reader, we have provided in [1] details that show how the notions presented here are represented in our Coq formalization, and one can convince oneself of the correctness of the results by simply compiling the accompanying Coq formalization [1] without having to read the proofs presented here. Additionally, a demonstration is included in the supplementary materials; one needs an OCaml compiler to be able to run the demonstration.

In Section 2 we present the various notions and definitions that are needed for our work. Section 3 has the three natural properties stated formally. In Sections 4-7, we prove the above mentioned results: Maximum matching, Local Uniqueness, Global Uniqueness, and Correctness of `Process_instruction` (partly moved to Appendix A. In Section 8 we describe how to build automated checkers. We also include a demonstration to show how our checker works on real data. Finally, the last section concludes the paper and describes future directions.

2 Preliminaries

We begin by introducing the various definitions that are needed for establishing our results. For ease of readability, we extensively use sets⁴ in our presentation. In the Coq formalization, however, we use lists instead. The choice of lists in the formalization allows us to use existing libraries that were developed in [13] and [8] for the purposes of modeling auctions, and, more importantly, it helps us in optimizing our algorithm leading to a reasonably fast OCaml program.⁵ In [1], we include detailed definitions and the main results alongside their formal versions, showing a faithful correspondence between the results and the formalization.

2.1 Orders

To avoid proving common properties of bids and asks twice, we model both as orders. An order is a 4-tuple $(id, timestamp, quantity, price)$. For an order ω , its components are $id(\omega)$, $timestamp(\omega)$, $qty(\omega)$, and $price(\omega)$, each of which is a natural number, and $qty(\omega) > 0$.

For a set of orders Ω , $ids(\Omega)$ represents the set of ids of the orders in Ω . For a set of orders Ω with distinct ids and an order $\omega \in \Omega$ such that $id(\omega) = id$, with slight abuse of notation, we define $timestamp(\Omega, id) = timestamp(\omega)$, $qty(\Omega, id) = qty(\omega)$, and $price(\Omega, id) = price(\omega)$.

We will often have a universe from which the bids and asks arise, which we call an order-domain. (B, A) is an **order-domain**, if B and A are sets of orders. Here, B represents a set of bids and A represents a set of asks. An order-domain where each id is distinct and each timestamp is distinct is called **admissible**.

⁴ Sets introduced in this work are all finite.

⁵ Using other data structures might yield further improvements.

We now define the terms “tradable” and “matchable”. Given two orders b (bid) and a (ask), we say b and a are **tradable** if $\text{price}(b) \geq \text{price}(a)$. An order-domain is **matchable** if it contains a bid and an ask that are tradable.

Next, we define competitiveness. A bid b_1 is more **competitive** compared to another bid b_2 , denoted by $b_1 \succ b_2$, if $\text{price}(b_1) > \text{price}(b_2)$ OR $(\text{price}(b_1) = \text{price}(b_2)$ AND $\text{timestamp}(b_1) < \text{timestamp}(b_2)$). Similarly, an ask a_1 is considered more **competitive** compared to another ask a_2 , denoted by $a_1 \succ a_2$, if $\text{price}(a_1) < \text{price}(a_2)$ OR $(\text{price}(a_1) = \text{price}(a_2)$ AND $\text{timestamp}(a_1) < \text{timestamp}(a_2)$).

We treat a set of orders also as a multiset where we suppress the quantity field of each order and set its multiplicity equal to its quantity. This view will help us succinctly state the **conservation** property and ease the formalization substantially. For a set of orders S_1 and S_2 , $S_1 \setminus S_2$, represents the usual set difference, whereas $S_1 - S_2$ represents the usual multiset difference between the two sets.

2.2 Transactions and matchings

For our purposes, keeping the price and timestamp in a transaction is redundant, as they can be derived from the participating bid and ask. Based on the application, a transaction between tradable orders b and a can be assigned an appropriate transaction price in the interval $[\text{price}(a), \text{price}(b)]$. For example, for the sake of concreteness, one may assume that such a transaction has price $\text{price}(a)$ and timestamp $\max\{\text{timestamp}(b), \text{timestamp}(a)\}$.

A transaction is a 3-tuple $(id_b, id_a, \text{quantity})$ of natural numbers where id_b and id_a represent the ids of the participating bid and ask, respectively, and $\text{quantity} > 0$. For a transaction t its components are represented by $\text{id}_{\text{bid}}(t)$, $\text{id}_{\text{ask}}(t)$, and $\text{qty}(t)$.

Let T be a set of transactions. We define $\text{ids}_{\text{bid}}(T)$ and $\text{ids}_{\text{ask}}(T)$ to be the sets of ids of the participating bids and asks, respectively. Next, we define $\text{Qty}_{\text{bid}}(T, id_b)$ to be sum of the transaction quantities of all transactions in T whose bid id is id_b , and $\text{Qty}_{\text{ask}}(T, id_a)$ to be the sum of the transaction quantities of all transactions in T whose ask id is id_a . For ease of readability, we often just use Qty instead of Qty_{ask} and Qty_{bid} . We define $\text{Vol}(T)$ to be the sum of the transaction quantities of all transactions in T .

We say a transaction t is **over** the order-domain (B, A) iff $\text{id}_{\text{bid}}(t) = \text{id}(b)$ for some $b \in B$ and $\text{id}_{\text{ask}}(t) = \text{id}(a)$ for some $a \in A$. We say that a transaction t is **valid** w.r.t order-domain (B, A) iff there exists $b \in B$ and $a \in A$ such that (i) $\text{id}_{\text{bid}}(t) = \text{id}(b)$ and $\text{id}_{\text{ask}}(t) = \text{id}(a)$, (ii) b and a are tradable, and (iii) $\text{qty}(t) \leq \min(\text{qty}(b), \text{qty}(a))$. We say that a set of transactions T is **valid** over an order-domain (B, A) if each transaction in T is valid over (B, A) .

Given a set of transactions, we would like to extract out the set of “traded” bids/asks. To this end, we define the functions **Bids** and **Asks** as follows. Let T be a set of transactions over an admissible order-domain (B, A) . We define $\text{Bids}(T, B)$ to be the set of all the bids in B that participate in T where the quantity of each bid b is set to the sum of the transaction quantities of the transactions in T involving b . Similarly, $\text{Asks}(T, A)$ is the set of asks in A that participate in T where the quantity of each ask is set to its total traded quantity in T . We often simply write $\text{Bids}(T)$ and $\text{Asks}(T)$ instead of $\text{Bids}(T, B)$ and $\text{Asks}(T, A)$ whenever B and A are clear from the context.

Finally, we define a matching, which is a set of transactions that can simultaneously arise from an order-domain. We also define the canonical form of a set of transactions, which will be often applied to matchings. We say a set of valid transactions M over an admissible order-domain (B, A) is a **matching** over (B, A) if for each order $\omega \in B \cup A$, $\text{Qty}(M, \text{id}(\omega)) \leq \text{qty}(\omega)$. We define the **canonical form** of a set of transactions M , denoted by $\mathcal{C}(M)$ to be a set of transactions satisfying the following two properties. For each bid-ask

pair (b, a) , $\mathcal{C}(M)$ consists of at most one transaction involving b and a . Furthermore, for each bid-ask pair (b, a) , the total transaction quantity between them in M is equal to the total transaction quantity between them in $\mathcal{C}(M)$. It is easy to see that $\mathcal{C}(M)$ always exists.

2.3 Order book and process

We now formally define instructions and order books. Buy, Sell, and Del are called **commands**. An instruction is a pair (Δ, ω) where Δ is a command and ω is an order. For convenience we represent an instruction (Δ, ω) by $\Delta \omega$. Also sometimes we represent (Del, ω) instruction simply by $\text{Del id}(\omega)$; this is done because for a Del command, only the id of the order matters. An **order book** is a list where each entry is an instruction.

We now define a “structured” order book, which satisfies two conditions. Firstly, the timestamps of the orders must be increasing. Secondly, the ids of the orders in the non-Del instructions in the order book must be all distinct. We will relax the second condition slightly which will help us in certain applications. We allow an order to have an id identical only to the id appearing in an immediately preceding Del order instruction. This will later help us in our application to implement an “update” instruction, by replacing it with a delete instruction followed by a Buy or Sell instruction carrying the same id. Formally, an order book $\mathcal{I} = [(\Delta_0, \omega_0), \dots, (\Delta_n, \omega_n)]$ is called **structured** if the following conditions hold.

- For all $i \in \{0, \dots, n-1\}$, $\text{timestamp}(\omega_i) < \text{timestamp}(\omega_{i+1})$.
- For all $i \in \{0, 1, \dots, n\}$, at least one of the following three conditions hold. (i) $\Delta_i = \text{Del}$. (ii) $\text{id}(\omega_i) \notin \text{ids}\{\omega_0, \dots, \omega_{i-1}\}$. (iii) $\text{id}(\omega_i) = \text{id}(\omega_{i-1})$ and $\Delta_{i-1} = \text{Del}$.

We now define a process, which represents an abstract online algorithm that will be fed resident bids and asks and an instruction and it will output a set of transactions and resulting resident bids and asks. A **process** is a function $(B, A, \tau) \mapsto (B', A', M)$ that takes as input sets of orders B, A , and an instruction τ and outputs sets of orders B', A' , and a set of transactions M .

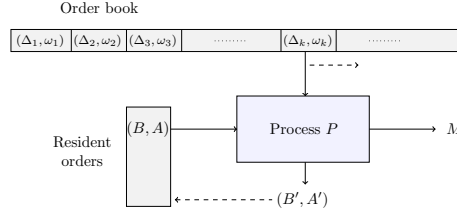
The input to a process is an order-domain, which represents the resident orders in the system, and an instruction. If this instruction is a delete id instruction, then the process is supposed to delete all resident orders with that id, and the “effective” order-domain potentially gets reduced. Otherwise, if the instruction is a buy/sell order, then the “effective” order-domain needs to include that order. We define **Absorb** that takes an order-domain and an instruction as input and outputs the “effective” order-domain. For an order-domain (B, A) and an instruction τ we define

$$\text{Absorb}(B, A, \tau) := \begin{cases} (\{\beta \in B \mid \text{id}(\beta) \neq \text{id}\}, \{\alpha \in A \mid \text{id}(\alpha) \neq \text{id}\}) & \text{if } \tau = \text{Del id} \\ (B \cup \{\beta\}, A) & \text{if } \tau = \text{Buy } \beta \\ (B, A \cup \{\alpha\}) & \text{if } \tau = \text{Sell } \alpha. \end{cases}$$

Observe that the following propositions follow immediately from the definition of **Absorb**.

► **Proposition 1.** *If τ is an instruction and (B, A) is an order-domain such that the timestamps of the orders in $B \cup A$ are all distinct and different from the timestamp appearing in τ and if $(B', A') = \text{Absorb}(B, A, \tau)$, then the timestamps of the orders in $B' \cup A'$ are all distinct.*

► **Proposition 2.** *If (B, A) is an order-domain such that the ids of the orders in $B \cup A$ are all distinct, then for all Del instructions τ , if $(B', A') = \text{Absorb}(B, A, \tau)$, then the ids of the orders in $B' \cup A'$ are all distinct.*



■ **Figure 1** Illustration of a Process: at time k the process P takes as input resident orders (B, A) and an instruction (Δ_k, ω_k) and outputs a matching M and a pair of sets of orders (B', A') that will act as resident orders for the next time step. **Iterated** takes as input a process P , an order book I , and a time k and outputs what P would output on I at time k . Thus, for the example in the figure, we have $\text{Iterated}(P, I, k) = (B', A', M)$.

► **Proposition 3.** *If τ is an instruction and (B, A) is an order-domain such that the ids of the orders in $B \cup A$ are all distinct and different from the id appearing in τ and if $(B', A') = \text{Absorb}(B, A, \tau)$, then the ids of the orders in $B' \cup A'$ are all distinct.*

To a process, we will usually feed inputs that satisfy certain properties, and such inputs we refer to as legal-inputs and are defined as follows. We say an order-domain (B, A) and an instruction τ forms a **legal-input** if B and A are not matchable and τ is such that $(B', A') = \text{Absorb}(B, A, \tau)$ is an admissible order-domain.

Finally, we define **Iterated**. Given a process P , an order book \mathcal{I} and a natural number k , we define $\text{Iterated}(P, \mathcal{I}, k)$ to be the output of P at time k when it is iteratively run on the order book \mathcal{I} . When $k > \text{length}(\mathcal{I})$, $\text{Iterated}(P, \mathcal{I}, k)$ returns $(\emptyset, \emptyset, \emptyset)$. Otherwise, $\text{Iterated}(P, \mathcal{I}, k)$ can be computed recursively as per the following algorithm.

■ **Algorithm 1** Iteratively running a process on an order book.

```

function Iterated(Process P, Order-book  $\mathcal{I}$ , natural number  $k$ )
  if  $k = 0$  or  $k > \text{length}(\mathcal{I})$  then return  $(\emptyset, \emptyset, \emptyset)$ 
   $(B, A, M) \leftarrow \text{Iterated}(P, \mathcal{I}, k - 1)$ 
  ▷  $B$  &  $A$  are resident orders &  $M$  is the matching outputted at time  $k - 1$ .
   $\tau \leftarrow k^{\text{th}}$  instruction in  $\mathcal{I}$ 
  return  $P(B, A, \tau)$ 

```

3 Three natural properties of a process

Having set up the definitions, we will now state the three natural properties formally.

We say a process P satisfies **positive bid-ask spread**, **price-time priority**, and **conservation** if for all order-domains (B, A) and an instruction τ such that (B, A) and τ forms a legal-input, $P(B, A, \tau) = (\hat{B}, \hat{A}, M)$ and $(B', A') = \text{Absorb}(B, A, \tau)$ implies the following three conditions.

1. **Positive Bid-Ask Spread:** \hat{B} and \hat{A} are not matchable.
2. **Price-Time Priority:** If a less competitive order ω gets traded in M , then all orders that are more competitive than ω must be fully traded in M . Formally,

- a. $\forall a, a' \in A', a \succ a'$ and $\text{id}(a') \in \text{ids}_{\text{ask}}(M) \implies \text{Qty}(M, \text{id}(a)) = \text{qty}(a)$
- b. $\forall b, b' \in B', b \succ b'$ and $\text{id}(b') \in \text{ids}_{\text{bid}}(M) \implies \text{Qty}(M, \text{id}(b)) = \text{qty}(b)$.

3. **Conservation:** P does not lose or add orders arbitrarily. For this, we have the following technical conditions.

- a. M is a matching over the order-domain (B', A')
- b. $\hat{B} = B' - \text{Bids}(M, B')$ c. $\hat{A} = A' - \text{Asks}(M, A')$.

The above definition appears in our Coq formalization as follows.

```

not (matchable hat_B hat_A).

forall b b', (In b B) /\ (In b' B) /\ (bcompetitive b b' /\ -eqcompetitive b b') /\ (In (id b') (ids_bid_aux M))
-> (Qty_bid M (id b)) = (oquantity b).

forall a a', (In a A) /\ (In a' A) /\ (acompetitive a a' /\ -eqcompetitive a a') /\ (In (id a') (ids_ask_aux M))
-> (Qty_ask M (id a)) = (oquantity a).

Matching M B' A'.
hat_B == (odiff B' (bids M B')).
hat_A == (odiff A' (asks M A')).

```

4 Maximum matching

In this section, we prove the maximum matching lemma.

► **Lemma 4** (Maximum matching). *Let P be a process that satisfies positive bid-ask spread and conservation. For all order-domain and instruction pairs $((B, A), \tau)$ that form legal-inputs, if $P(B, A, \tau) = (\hat{B}, \hat{A}, M)$, then for all matchings M' over $\text{Absorb}(B, A, \tau)$, $\text{Vol}(M) \geq \text{Vol}(M')$.*

Proof of Lemma 4. Let us fix a process P , an order-domain (B, A) , and an instruction τ as in the lemma statement. Let $(\hat{B}, \hat{A}, M) = P(B, A, \tau)$ and $(B', A') = \text{Absorb}(B, A, \tau)$. Let M' be an arbitrary matching over (B', A') . We need to show $\text{Vol}(M) \geq \text{Vol}(M')$.

Now we have the following three cases depending on τ .

Case: τ is Del id . In this case $B' = \{b \in B \mid \text{id}(b) \neq id\} \subseteq B$ and $A' = \{a \in A \mid \text{id}(a) \neq id\} \subseteq A$ as $(B', A') = \text{Absorb}(B, A, \tau)$.

Since (B, A) and τ forms a legal input, (B, A) is not matchable. Consequently, $(B', A') = \text{Absorb}(B, A, \tau)$ is also not matchable as $B' \subseteq B$ and $A' \subseteq A$. Thus, no valid transaction over (B', A') exists. Since the matching M' is a set of valid transactions over (B', A') , $M' = \emptyset$. Thus, $\text{Vol}(M') = 0$, and we are trivially done.

Case: τ is Buy β . In this case $B' = B \cup \{\beta\}$ and $A' = A$ as $(B', A') = \text{Absorb}(B, A, \tau)$.

If (B', A') is not matchable, then as in the previous case, $\text{Vol}(M') = 0$, and we are trivially done. Thus, we may assume (B', A') is matchable. Now since (B, A) is not matchable, only $\beta \in B'$ is tradable with some ask in A' . Thus, for all valid transactions t over (B', A') , $\text{id}_{\text{bid}}(t) = \text{id}(\beta)$.

We assume for contradiction: $\text{Vol}(M) < \text{Vol}(M')$. We will show that there exists a bid $\hat{b} \in \hat{B}$ and an ask $\hat{a} \in \hat{A}$ that remain resident and are tradable, contradicting positive bid-ask spread.

Since M and M' are matchings over (B', A') consisting of only valid transactions t such that $\text{id}_{\text{bid}}(t) = \text{id}(\beta)$ and $\text{Vol}(M) < \text{Vol}(M')$, there must exist an ask $a \in A'$ which is tradable with β , such that its trade quantity in M is strictly less than its trade quantity in M' , i.e.,

$$\text{Qty}_{\text{ask}}(M, \text{id}(a)) < \text{Qty}_{\text{ask}}(M', \text{id}(a)) \leq \text{qty}(a),$$

and β and a are tradable. Thus, from **conservation** part c, a part of a will remain resident in \hat{A} . In particular, there exists $\hat{a} \in \hat{A}$ such that $\text{price}(\hat{a}) = \text{price}(a)$. Now, since β and a are tradable, so are β and \hat{a} .

Again since M' is a matching over (B', A') consisting of only valid transactions t such that $\text{id}_{\text{bid}}(t) = \text{id}(\beta)$, $\text{Vol}(M') \leq \text{qty}(\beta)$. Thus, from our assumption, $\text{Vol}(M) < \text{Vol}(M') \leq \text{qty}(\beta)$ which implies, from **conservation** part b, some part of β remains untraded in M . In particular, there exists $\hat{\beta} \in \hat{B}$ such that $\text{price}(\hat{\beta}) = \text{price}(\beta)$. Now since β and \hat{a} are tradable, $\hat{\beta}$ and \hat{a} are tradable, which contradicts **positive bid-ask spread**, completing the proof.

Case: τ is Sell α . The proof in this case is symmetric to the above case. \blacktriangleleft

5 Local Uniqueness

In this section, we prove the local uniqueness theorem.

► Theorem 5. *Let P_1 and P_2 be processes that satisfy **price-time priority**, **positive bid-ask spread**, and **conservation**. For all order-domain instruction pairs $((B, A), (\Delta, \omega))$ that form legal-inputs, if for each $i \in \{1, 2\}$, $P_i(B, A, (\Delta, \omega)) = (\hat{B}_i, \hat{A}_i, M_i)$, then $(\hat{B}_1, \hat{A}_1, \mathcal{C}(M_1)) = (\hat{B}_2, \hat{A}_2, \mathcal{C}(M_2))$. Furthermore, the following statements hold for each i .*

- (1) \hat{B}_i and \hat{A}_i are not matchable.
- (2) The timestamps of orders in \hat{B}_i and \hat{A}_i are distinct and form a subset of the set of timestamps of the orders in $B \cup A \cup \{\text{timestamp}(\omega)\}$.
- (3) The ids of orders in \hat{B}_i and \hat{A}_i are distinct and form a subset of $\text{ids}(B \cup A) \cup \{\text{id}(\omega)\}$.
- (4) $\Delta = \text{Del} \implies \text{id}(\omega) \notin \text{ids}(\hat{B}_i \cup \hat{A}_i)$.

Proof. Fix processes P_1 and P_2 that satisfy the three properties. Fix an order-domain (B, A) and an instruction (Δ, ω) which forms a legal-input. Let for each $i \in \{1, 2\}$ $(\hat{B}_i, \hat{A}_i, M_i) = P_i(B, A, (\Delta, \omega))$. Let $(B', A') = \text{Absorb}(B, A, (\Delta, \omega))$.

We will show $(\hat{B}_1, \hat{A}_1, \mathcal{C}(M_1)) = (\hat{B}_2, \hat{A}_2, \mathcal{C}(M_2))$.

Proofs of (1)-(4) above are relatively straightforward, and we omit detailed proofs: (1) is precisely **positive bid-ask spread**. (2) and (3) follow from the fact that timestamps and ids of the resident orders arise from the timestamps of the orders in the order-domain (B', A') , which is admissible. (4) follows from the fact that if $\Delta = \text{Del}$, then $\text{id}(\omega) \notin \text{ids}(A' \cup B')$.

To show $(B_1, A_1, \mathcal{C}(M_1)) = (B_2, A_2, \mathcal{C}(M_2))$, we will do a case analysis based on Δ .

First note that both M_1 and M_2 are maximum volume matching between B' and A' from Lemma 4.

Case 1: Δ is Del. In this case $B' \subseteq B$ and $A' \subseteq A$ as $(B', A') = \text{Absorb}(B, A, (\Delta, \omega))$. Now, since (B, A) is not matchable, (B', A') is also not matchable, implying $M_1 = M_2 = \emptyset$ as no valid transactions exist over the order-domain (B', A') . From **conservation** parts b and c, we have $\hat{B}_i = B'$ and $\hat{A}_i = A'$ for each $i \in \{1, 2\}$, and we are done.

Case 2: Δ is Buy. In this case $B' = B \cup \{\omega\}$ and $A' = A$ as $(B', A') = \text{Absorb}(B, A, (\Delta, \omega))$. Also since (B, A) is not matchable, then every valid transaction t over the order-domain (B', A') is such that $\text{id}_{\text{bid}}(t) = \text{id}(\omega)$.

If ω is not tradable with any ask in A' , then we are done like in the previous case as there are no possible valid transactions over (B', A') , implying $M_1 = M_2 = \emptyset$. And as before $B_i = B'$ and $A_i = A'$ for each $i \in \{1, 2\}$ follows from **conservation**, and we are done.

Thus, we may assume ω is tradable with some ask in A' . We will first show that for each ask $a \in A$, the total traded volume of a in M_1 is equal to the total traded volume of a in M_2 . Assume for the sake of contradiction, there exists an ask a that has more traded quantity in M_1 than in M_2 . This implies a is partially traded in M_2 . Since trade volumes of both M_1 and M_2 are equal (as they are both maximum), there must be an ask a' such that total traded volume of a' in M_2 is more than that in M_1 . In particular, this implies that a' has at least one transaction in M_2 .

Now either $a \succ a'$ or $a' \succ a$. First assume $a \succ a'$. The fact that a more competitive order a is partially traded in M_2 while a less competitive order a' is being traded contradicts **price-time priority**. Next assume $a' \succ a$, the fact that in M_1 , a' is partially traded and a is also traded, leads to a similar contradiction. Thus, for all $a \in A'$, $\text{Qty}_{\text{ask}}(M_1, \text{id}(a)) = \text{Qty}_{\text{ask}}(M_2, \text{id}(a))$.

Now recall in each valid transaction t over the order-domain (B', A') $\text{id}_{\text{bid}}(t) = \text{id}(\omega)$. As M_i consists of only such transactions t , $\mathcal{C}(M_i)$ consists of transactions of the form $(\text{id}(\omega), \text{id}(a), q)$ where $q = \text{Qty}_{\text{ask}}(M_i, \text{id}(a))$ and $a \in A'$. Now since for all $a \in A'$, $\text{Qty}_{\text{ask}}(M_1, \text{id}(a)) = \text{Qty}_{\text{ask}}(M_2, \text{id}(a))$, a transaction is in $\mathcal{C}(M_1)$ iff it is in $\mathcal{C}(M_2)$, proving $\mathcal{C}(M_1) = \mathcal{C}(M_2)$.

Now we prove $\hat{A}_1 = \hat{A}_2$. Fix an $i \in \{1, 2\}$. It follows from **conservation** part c that each $a \in A'$ which is not fully traded in M_i by P_i will have a corresponding $\hat{a}_i \in \hat{A}_i$ such that $\text{qty}(\hat{a}_i) = \text{qty}(a) - \text{Qty}_{\text{ask}}(M_i, \text{id}(a))$ and $(\text{id}(\hat{a}), \text{timestamp}(\hat{a}), \text{price}(\hat{a})) = (\text{id}(a), \text{timestamp}(a), \text{price}(a))$. Furthermore, no element exists in \hat{A}_i whose **id**, **timestamp**, and **price** do not match with the respective attributes of some element in A' . Now, we proved earlier that for all $a \in A'$, $\text{Qty}_{\text{ask}}(M_1, \text{id}(a)) = \text{Qty}_{\text{ask}}(M_2, \text{id}(a))$. Combining these facts, we get $\hat{A}_1 = \hat{A}_2$.

$\hat{B}_1 = \hat{B}_2$ follows from **conservation** part b: Observe that if ω gets fully traded in M_1 and M_2 , $\hat{B}_1 = \hat{B}_2 = B$, as all elements of B remain resident from **conservation**. If ω gets partially traded in M_1 and M_2 , then $\hat{B}_1 = \hat{B}_2 = B \cup \{\hat{\omega}\}$, where $(\text{id}(\hat{\omega}), \text{timestamp}(\hat{\omega}), \text{price}(\hat{\omega})) = (\text{id}(\omega), \text{timestamp}(\omega), \text{price}(\omega))$ and $q(\hat{\omega}) = \text{qty}(\omega) - \text{Vol}(M_i)$ (for each $i \in \{1, 2\}$).

Case 3: Δ is Sell. Here, the proof is symmetric to the proof in Case 2. ◀

6 Global uniqueness

In this section, we prove the global uniqueness theorem.

► **Theorem 6.** *Let P_1 and P_2 be processes that satisfy **positive bid-ask spread**, **price-time priority**, and **conservation**. Then, for all structured order books \mathcal{I} and natural numbers k if $\text{literated}(P_1, \mathcal{I}, k) = (B_1, A_1, M_1)$ and $\text{literated}(P_2, \mathcal{I}, k) = (B_2, A_2, M_2)$, then $(B_1, A_1, \mathcal{C}(M_1)) = (B_2, A_2, \mathcal{C}(M_2))$.*

In our Coq formalization, this theorem appears as follows.

```
(Properties P1) /\ (Properties P2) /\ structured I ->
(cform (Mlist (iterated P1 I k))) === (cform (Mlist (iterated P2 I k))) /\
(Blist (iterated P1 I k)) === (Blist (iterated P2 I k)) /\
(Alist (iterated P1 I k)) === (Alist (iterated P2 I k)).
```

To prove the above theorem we lift the local uniqueness theorem, Theorem 5, using induction. We achieve this by strengthening the theorem statement that provides us with a strong enough induction hypothesis for establishing the pre-conditions needed to apply Theorem 5.

Proof of Theorem 6. Fix processes P_1 and P_2 and a structured order book $\mathcal{I} = [(\Delta_0, \omega_0), \dots, \Delta_n, \omega_n]$, and let $\text{Iterated-P}_i(k)$ denote $\text{Iterated}(P_i, \mathcal{I}, k)$ for $i \in \{1, 2\}$.

Notice for $k > \text{length}(\mathcal{I}) = n + 1$, $\text{Iterated-P}_1(k) = \text{Iterated-P}_2(k) = (\emptyset, \emptyset, \emptyset)$ and we are trivially done. Now we will show the theorem statement holds for $k \leq n + 1$. More generally, we prove the following statement by induction on k .

For all $k \leq n + 1$, if $\text{Iterated-P}_1(k) = (B_1^k, A_1^k, M_1^k)$ and $\text{Iterated-P}_2(k) = (B_2^k, A_2^k, M_2^k)$, then the following properties hold.

- (1) $(B_1^k, A_1^k, \mathcal{C}(M_1^k)) = (B_2^k, A_2^k, \mathcal{C}(M_2^k))$.
- (2) B_1^k and A_1^k are not matchable.
- (3) The timestamps of orders in $B_1^k \cup A_1^k$ are distinct and form a subset of the set of timestamps of $\{\omega_0, \dots, \omega_{k-1}\}$.
- (4) Orders in $B_1^k \cup A_1^k$ have distinct ids and they belong to $\text{ids}(\{\omega_0, \dots, \omega_{k-1}\})$.
- (5) $k \geq 1$ and $\Delta_{k-1} = \text{Del} \implies \text{id}(\omega_{k-1}) \notin \text{ids}(B_1^k \cup A_1^k)$.

Observe that proving (1) above would complete the proof of Theorem 6.

Base case: When $k = 0$, $\text{Iterated-P}_1(0) = \text{Iterated-P}_2(0) = (\emptyset, \emptyset, \emptyset)$, and properties (1)–(5) trivially hold.

We now assume that the above statement (properties (1)–(5)) holds for $k = t$ and we will show that the above statement holds for $k = t + 1$. Notice for each $i \in \{1, 2\}$, $\text{Iterated-P}_i(t + 1)$ first computes $\text{Iterated-P}_i(t)$ recursively to obtain (B_i^t, A_i^t) and then returns $P_i(B_i^t, A_i^t, (\Delta_t, \omega_t)) = (B_i^{t+1}, A_i^{t+1}, M_i^{t+1})$. Our proof proceeds in two parts. First, using the induction hypothesis we prove that $(B_i^t, A_i^t, (\Delta_t, \omega_t))$ forms a legal input, which is the pre-condition to apply Theorem 5. Next, we invoke Theorem 5 and establish that properties (1)–(5) hold for $k = t + 1$.

First part. $(B_i^t, A_i^t, (\Delta_t, \omega_t))$ forms a legal input. To show this, observe from the definition of legal input, we need to show (i) B_i^t and A_i^t are not matchable and (ii) $\text{Absorb}(B_i^t, A_i^t, (\Delta_t, \omega_t))$ forms an admissible order-domain. (i) is immediate from the induction hypothesis (property (2)). To show (ii) we need to show that if $\text{Absorb}(B_i^t, A_i^t, (\Delta_t, \omega_t)) = (B', A')$, then the timestamps of the orders in $B' \cup A'$ are all distinct and the ids of the orders in $B' \cup A'$ are all distinct. The timestamps of the orders in $B' \cup A'$ are distinct follows from Proposition 1, since from the induction hypothesis (property (3)) it follows that the orders in B_1^t and A_1^t have distinct timestamps from the set $\{\text{timestamp}(\omega_0), \dots, \text{timestamp}(\omega_{t-1})\}$, and they are strictly smaller than $\text{timestamp}(\omega_t)$ as \mathcal{I} is structured. Below we show that the ids of $B' \cup A'$ are distinct by considering two cases: $\Delta_t = \text{Del}$ and $\Delta_t \neq \text{Del}$.

- Case: $\Delta_t = \text{Del}$. We are immediately done from Proposition 2, since orders in $B_1^t \cup A_1^t$ have distinct ids from the induction hypothesis (property (4)).
- Case: $\Delta_t \neq \text{Del}$. Since $\Delta_t \neq \text{Del}$ and \mathcal{I} is structured, either (a) $\text{id}(\omega_t) \notin \text{ids}\{\omega_0, \dots, \omega_{t-1}\}$ or (b) $\text{id}(\omega_t) = \text{id}(\omega_{t-1})$ and $\Delta_{t-1} = \text{Del}$. In either case we claim that $\text{id}(\omega_t) \notin \text{ids}(A_1^t \cup B_1^t)$. From the induction hypothesis (property (4)), we know orders in $B_1^t \cup A_1^t$ have distinct ids from the set $\text{ids}\{\omega_0, \dots, \omega_{t-1}\}$. So in case (a), it immediately follows that $\text{id}(\omega_t) \notin \text{ids}(A_1^t \cup B_1^t)$. In case (b), since $\Delta_t = \text{Del}$, from induction hypothesis (property (5)), we have that $\text{id}(\omega_{t-1}) \notin \text{ids}(B_1^t \cup A_1^t)$. In case (b) we also have $\text{id}(\omega_t) = \text{id}(\omega_{t-1})$. Thus, combining these two facts, we have $\text{id}(\omega_t) \notin \text{ids}(B_1^t \cup A_1^t)$. Now using the claim and the fact that ids of orders in $B_1^t \cup A_1^t$ are all distinct, we invoke Proposition 3 to get that ids of orders in $B' \cup A'$ are all distinct.

Second part. properties (1)-(5) hold for $k = t + 1$. (1) holds: From the induction hypothesis (property (1)), we have $(B_1^k, A_1^k) = (B_2^k, A_2^k)$. Now since $(B_1^k, A_1^k, (\Delta_t, \omega_t))$ forms a legal input, we are immediately done by invoking Theorem 5. (2) and (5) also immediately follow by invoking Theorem 5. (4) holds: Theorem 5 implies that the ids of orders in $B_1^{t+1} \cup A_1^{t+1}$ are all distinct and form a subset of $(\text{ids}(B_1^t \cup A_1^t) \cup \{\text{id}(\omega_t)\})$. Now from induction hypothesis (property (4)), we have $\text{ids}(B_1^t \cup A_1^t) \subseteq \text{ids}(\{\omega_0, \dots, \omega_{t-1}\})$. Thus, ids of the orders in $B_1^{t+1} \cup A_1^{t+1}$ forms a subset of $\text{ids}(\{\omega_0, \dots, \omega_t\})$, and we are done. Using an almost identical argument, we can show that (3) holds. ◀

7 Verified Algorithm

Here we introduce a natural algorithm for continuous double auctions.

Algorithm 2 Process for continuous market.

```

function PROCESS_INSTRUCTION(Bids  $B$ , Asks  $A$ , Instruction  $\tau$ )
  if  $\tau = \text{Del } id$  then Del_order( $B, A, id$ )
  if  $\tau = \text{Buy } \beta$  then Match_bid( $B, A, \beta$ )
  if  $\tau = \text{Sell } \alpha$  then Match_ask( $B, A, \alpha$ )

```

In the Coq formalization of Process_instruction, we sort the list of asks and bids by their competitiveness before calling a subroutine; as a result, the most competitive bid and ask are on top of their respective lists.

Algorithm 3 Matching an ask.

```

function MATCH_ASK(Bids  $B$ , Asks  $A$ , order  $\alpha$ )           ▷  $\alpha$  is an ask.
  if  $B = \emptyset$  then return ( $B, A \cup \{\alpha\}, \emptyset$ )
   $\beta \leftarrow \text{Extract\_most\_competitive}(B)$            ▷ Note:  $B \leftarrow B \setminus \{\beta\}$ .
  if  $\text{price}(\beta) < \text{price}(\alpha)$  then return ( $B \cup \{\beta\}, A \cup \{\alpha\}, \emptyset$ )
  ▷ From now on  $\beta$  and  $\alpha$  are tradable.
  if  $\text{qty}(\beta) = \text{qty}(\alpha)$  then  $m \leftarrow (\text{id}(\beta), \text{id}(\alpha), \text{qty}(\alpha))$ 
    return ( $B, A, \{m\}$ )
  if  $\text{qty}(\beta) > \text{qty}(\alpha)$  then  $m \leftarrow (\text{id}(\beta), \text{id}(\alpha), \text{qty}(\alpha))$ 
     $B' \leftarrow B \cup \{(\text{id}(\beta), \text{timestamp}(\beta), \text{qty}(\beta) - \text{qty}(\alpha), \text{price}(\beta))\}$ 
    return ( $B', A, \{m\}$ )
  if  $\text{qty}(\beta) < \text{qty}(\alpha)$  then  $m \leftarrow (\text{id}(\beta), \text{id}(\alpha), \text{qty}(\beta))$ 
     $\alpha' \leftarrow (\text{id}(\alpha), \text{timestamp}(\alpha), \text{qty}(\alpha) - \text{qty}(\beta), \text{price}(\alpha))$ 
     $(B', A', M') \leftarrow \text{Match\_ask}(B, A, \alpha')$ 
     $M \leftarrow M' \cup \{m\}$ 
    return ( $B', A', M$ )

```

The Match_Bid subroutine is symmetric to the Match_Ask subroutine and we do not present it explicitly here.

Algorithm 4 Deleting an order.

```

function DEL_ORDER( $B, A, id$ )
  if  $id \in \text{ids}(B)$  then  $B \leftarrow \text{remove}(B, id)$ 
  if  $id \in \text{ids}(A)$  then  $A \leftarrow \text{remove}(A, id)$ 
  return ( $B, A, \emptyset$ )

```

Next, we show that the above algorithm satisfies the three natural properties.

► **Theorem 7.** *Process_instruction satisfies positive bid-ask spread, price-time priority, and conservation.*

In our Coq formalization, the above theorem statement appears as follows.

```
Properties Process_instruction.
```

The proof of this result is outlined in Appendix A.

8 Application: checker

In this section, we discuss how we use our verified algorithm and the global uniqueness theorem to design an automated checker that detects errors in exchange algorithms that run continuous double auctions from their trade logs. We implement such a checker and run it on trade logs from an exchange. We include, as part of the supplementary materials accompanying this paper, the OCaml source code of our checker, trade logs for two example stocks (created from real stocks from a stock exchange after appropriate preprocessing and masking), and a shell script that compiles our code and runs it on the trade logs of the two stocks.

As discussed in the paper, exchanges for each traded product maintain an order book that contains all the incoming orders and a corresponding trade book that contains all the transactions that are generated as trade logs. These trade logs are accessible to the regulators. We show a market regulator can use these trade logs to automatically determine whether an exchange is complying with the three natural requirements of **price-time priority**, **positive bid-ask spread**, and **conservation** while generating transactions. Recall, as implied by our global uniqueness theorem, these three properties completely specify continuous double auctions.

8.1 Algorithm for checker

Given an order book and the corresponding trade book, the checker first runs the verified matching algorithm on the order book to generate “verified” matchings. Then, the canonical forms of the matchings in the trade book with the verified matchings are compared, one time-step at a time. If for any time-step, the canonical forms of the matchings do not match, the checker outputs a mismatch detected message along with the corresponding matchings. Otherwise, if all matchings match, the checker terminates without a mismatch message.

For a given order book and trade book, if the above checker finds a mismatch, then we can conclude, from the global uniqueness theorem, that the exchange algorithm violates at least one of the three properties. Otherwise, at least for the instance at hand, the exchange algorithm output is as good as that of a verified algorithm.

In our implementation of the checker, the verified matching algorithm and the canonical form functions are directly extracted from our formalization using Coq’s code extraction feature and used as subroutines.

8.2 Preprocessing an order book

Real exchanges implement more complex instruction types than our three primitives: buy/sell/delete. Here we briefly discuss some of these instruction types and how to convert them into primitive types in a preprocessing step.

- Updates. A buyer or seller might want to update the quantity or price of his order using an update order instruction. For our test exchange, the rule is if the quantity of the order decreases, then the timestamp of the original order is retained. Otherwise, if the

quantity is increased or the price changes the new order carries the timestamp of the update instruction. Our test exchange as part of the update instruction also maintains the information of the quantity of the order that remains untraded. Updates can be easily implemented by replacing it with a delete instruction followed by a new buy/sell instruction with the updated attributes.

- Market orders. Market orders are orders that do not specify a limit price and are ready to be traded at any price. We can implement such an order quite easily by keeping the price 0 for a sell order and ∞ (in our implementation we use the maximum number supported by the system) for a buy order.
- Immediate or cancel orders (IOCs). An IOC is an order that needs to be immediately removed from the system after it is processed, i.e., it should never become a resident order. Such orders can be implemented by replacing them with a buy/sell instruction followed by a delete instruction.
- Stop-loss orders. Stop-loss orders are orders that are triggered when certain events happen, like when the price of a transaction goes below a threshold. For our test exchange, the timestamp of the triggering event is provided. Consequently, these orders can be treated as normal orders when inserted in the order book at a position corresponding to the timestamp of the triggering event.

Note that the preprocessing step can at the most double the number of instructions in the order book. Apart from these standard orders discussed above, certain exchanges allow for more complicated orders which include “iceberg orders”. Iceberg orders are rare and are more complicated to preprocess where the priority of the orders depends on factors that cannot be fully determined by just price and time. Although we can implement such orders through some preprocessing hacks, ideally the matching algorithm should be enriched to handle such orders.

8.3 Demonstration

We include, as part of the supplementary materials [1] accompanying this paper, the OCaml source code of our checker, trade logs for two example stocks (created from real data after appropriate preprocessing and masking), and a shell script that compiles our code and runs it on the trade logs of the two stocks. The checker takes under a second to process both the order books. For the first stock, the order book has about 16000 instructions; the checker detects that there is no mismatch. For the second stock, the order book has about 15000 instructions; the checker detects a mismatch. On inspection, we found that a delete instruction appears in the order book before the corresponding order was placed causing the mismatch.

We believe that it would be difficult to detect such rare anomalies without the systematic application of formal methods; in absence of formal specifications, uniqueness theorems, and verified algorithms, such anomalies can be easily brushed aside as complex and probably unavoidable program behavior.

9 Conclusions

In this work, we formally introduced continuous double auctions and specified them in terms of three natural and necessary properties. Then, we showed that a natural algorithm possesses these properties. Based on our work, we were able to produce a checker that automatically detects errors in existing exchange algorithms by going over their trade logs.

There are many opportunities in the field of certified auctions. For continuous double auctions alone, one could look at various sophisticated priority rules. For example, what would be the specifications for parity/priority matchings (as used by NYSE)? What would be the specifications for price-time priority matchings in the presence of iceberg orders? What would be the specifications for decentralized exchanges (for cryptocurrencies)?

With this work and previous works in the field, a new paradigm for formalizing various auction mechanisms is emerging, which we believe has useful policy implications for regulators. With the formal methods technology, it is now possible to formally verify systems and require auctioneers to implement them, and also check if the implementation is correct by checking the logs of the auctions that were conducted.

References

- 1 Supplementary materials: Coq formalization and demonstration, 2022. URL: <https://github.com/suneel-sarswat/cda>.
- 2 Iliano Cervesato, Sharjeel Khan, Giselle Reis, and Dragisa Žunić. Formalization of automated trading systems in a concurrent linear framework. In *Linearity-TLLA@FLoC*, volume 292 of *EPTCS*, pages 1–14, 2018. [arXiv:1904.06159](https://arxiv.org/abs/1904.06159).
- 3 Frankfurt Stock Exchange. Market Model for the Trading Venue Xetra. https://www.xetra.com/resource/blob/1963528/ceda0de49d5b8db5e33cdb17375e4cc9/data/T7_R.8.1_Market_Model-en.pdf, Sep 22, 2021.
- 4 Daniel Friedman. The double auction market institution: A survey. *The double auction market: Institutions, theories, and evidence*, 14:3–25, 1993.
- 5 Larry Harris. *Trading and exchanges: Market microstructure for practitioners*. OUP USA, 2003.
- 6 Cezary Kaliszzyk and Julian Parsert. Formal microeconomic foundations and the first welfare theorem. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 91–101. ACM, 2018. [doi:10.1145/3167100](https://doi.org/10.1145/3167100).
- 7 Vasilios Mavroudis and Hayden Melton. Libra: Fair order-matching for electronic financial exchanges. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 156–168, 2019.
- 8 Raja Natarajan, Suneel Sarswat, and Abhishek Kr Singh. Verified double sided auctions for financial markets. In Liron Cohen and Cezary Kaliszzyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPICs*, pages 28:1–28:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. [doi:10.4230/LIPICs.ITP.2021.28](https://doi.org/10.4230/LIPICs.ITP.2021.28).
- 9 Jinzhong Niu and Simon Parsons. Maximizing matching in double-sided auctions. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1283–1284, 2013.
- 10 Grant Passmore, Simon Cruanes, Denis Ignatovich, Dave Aitken, Matt Bray, Elijah Kagan, Kostya Kanishev, Ewen Maclean, and Nicola Mometto. The imandra automated reasoning system (system description). In *International Joint Conference on Automated Reasoning*, pages 464–471. Springer, 2020.
- 11 Grant Olney Passmore and Denis Ignatovich. Formal verification of financial algorithms. In *26th International Conference on Automated Deduction, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017.
- 12 Tobias Preis. Price-time priority and pro rata matching in an order book model of financial markets. In *Econophysics of Order-driven Markets*, pages 65–72. Springer, 2011.
- 13 Suneel Sarswat and Abhishek Kr Singh. Formally verified trades in financial markets. In *Formal Methods and Software Engineering – 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings*, volume 12531 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2020. [doi:10.1007/978-3-030-63406-3_13](https://doi.org/10.1007/978-3-030-63406-3_13).

- 14 Securities Exchange Board of India (SEBI). Order in the matter of NSE Colocation, Apr 30, 2019. Order in the matter of NSE Colocation.
- 15 The Coq Development Team. The coq reference manual, release 8.12.2, December 11 2020. URL: <https://github.com/coq/coq/releases/download/V8.12.2/coq-8.12.2-reference-manual.pdf>.
- 16 U.S. Securities and Exchange Commission (SEC). SEC Charges UBS Subsidiary With Disclosure Violations and Other Regulatory Failures in Operating Dark Pool. <https://www.sec.gov/news/pressrelease/2015-7.html>, July, 2015.
- 17 U.S. Securities and Exchange Commission (SEC). NYSE to Pay US Dollar 14 Million Penalty for Multiple Violations. <https://www.sec.gov/news/press-release/2018-31>, March 6, 2018.
- 18 U.S. Securities and Exchange Commission (SEC). SEC Charges NYSE for Repeated Failures to Operate in Accordance With Exchange Rules. <https://www.sec.gov/news/press-release/2014-87>, May 1, 2014.
- 19 Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM computing surveys (CSUR)*, 41(4):1–36, 2009.
- 20 Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: theory and implementation. *Decision Support Systems*, 24(1):17–27, 1998.
- 21 Junbeom Yoo, Taihyo Kim, Sungdeok Cha, Jang-Soo Lee, and Han Seong Son. A formal software requirements specification method for digital nuclear plant protection systems. *Journal of Systems and Software*, 74(1):73–83, 2005.
- 22 Dengji Zhao, Dongmo Zhang, Md Khan, and Laurent Perrussel. Maximal matching for double auction. In *Australasian Joint Conference on Artificial Intelligence*, pages 516–525. Springer, 2010.

A Correctness of Process_instruction

As Process_instruction on each input invokes one of Match_Ask, Match_Bid, and Del_Order, it is enough to show that these subroutines have the desired properties. It is relatively straightforward to show that Del_Order has the properties and the proof for Match_Bid and Match_Ask are completely symmetric. Thus, in our presentation, we just show that Match_Ask has the desired properties.

A.1 Match_Ask satisfies positive bid-ask spread

The Match_Ask subroutine satisfies positive bid-ask spread follows immediately from the following lemma.

► **Lemma 1.** *Let $(B, A \cup \{\alpha\})$ be an admissible order-domain. If $\text{Match_Ask}(B, A, \alpha) = (\hat{B}, \hat{A}, M)$ and (B, A) is not matchable, then (\hat{B}, \hat{A}) is not matchable.*

Proof. Fix an A and an α . We prove the lemma by induction on $|B|$.

Base case: trivial.

Induction hypothesis: For all $|B| < k$, If $(B, A \cup \{\alpha\})$ is an admissible domain, (B, A) is not matchable, and $\text{Match_Ask}(B, A, \alpha) = (\hat{B}, \hat{A}, M)$, then (\hat{B}, \hat{A}) is not matchable.

Induction step: We prove the statement when $|B| = k$.

Let β be the most competitive bid in B . If β is not tradable with α , then as per Match_Ask, $\hat{B} = B$ and $\hat{A} = A \cup \{\alpha\}$. Now since α is not tradable with the most competitive bid in B , it is not tradable with any bid in B . Coupled with the fact that (B, A) is not matchable, we get (\hat{B}, \hat{A}) is not matchable.

From now on, we may assume that β and α are tradable. We have three cases based on the relationship between $\text{qty}(\beta)$ and $\text{qty}(\alpha)$.

- If $\text{qty}(\beta) = \text{qty}(\alpha)$, then as per `Match_Ask`, $(\hat{B}, \hat{A}) = (B \setminus \{\beta\}, A)$, and we are trivially done as (B, A) is not matchable.
- If $\text{qty}(\beta) > \text{qty}(\alpha)$, then as per `Match_Ask`, $(\hat{B}, \hat{A}) = ((B \setminus \{\beta\}) \cup \{\beta'\}, A)$ where β and β' only differ in their quantities. Now since (B, A) is not matchable, (\hat{B}, \hat{A}) is not matchable, and we are done.
- If $\text{qty}(\beta) < \text{qty}(\alpha)$, then `Match_Ask` on (B, A, α) first recursively calls `Match_Ask` on $(B \setminus \{\beta\}, A, \alpha')$ where α' is obtained from α by reducing its quantity to $\text{qty}(\alpha) - \text{qty}(\beta)$. Let $(B', A', M') = \text{Match_Ask}(B \setminus \{\beta\}, A, \alpha)$. Then, `Match_Ask` returns $(\hat{B}, \hat{A}, M) = (B', A', M)$ where M is obtained from M' in some way. Now since, the set of bids in the recursive call has cardinality $|B \setminus \{\beta\}| = k - 1$, $(B \setminus \{\beta\}, A \cup \{\alpha'\})$ is admissible (as $(B, A \cup \{\alpha\})$ admissible), and $(B \setminus \{\beta\}, A)$ is not matchable (as (B, A) not matchable), we can apply the induction hypothesis to conclude (B', A') is not matchable. Now as $(\hat{B}, \hat{A}) = (B', A')$, we are done. ◀

A.2 Match_Ask satisfies price-time priority

The subroutine `Match_Ask` satisfies price-time priority follows immediately from the following lemma.

► **Lemma 2.** *Let $(B, A \cup \{\alpha\})$ be an admissible order-domain such that (B, A) is not matchable. If $\text{Match_Ask}(B, A, \alpha) = (\hat{B}, \hat{A}, M)$, then*

$$(i) \forall a, a' \in A \cup \{\alpha\}, a \succ a' \text{ and } \text{id}(a') \in \text{ids}_{\text{ask}}(M) \implies \text{Qty}(M, \text{id}(a)) = \text{qty}(a)$$

$$\text{and } (ii) \forall b, b' \in B, b \succ b' \text{ and } \text{id}(b') \in \text{ids}_{\text{bid}}(M) \implies \text{Qty}(M, \text{id}(b)) = \text{qty}(b).$$

Proof. We first show (i). If $M = \emptyset$, then price-time priority trivially holds. Otherwise, since (B, A) is not matchable, only α in $A \cup \{\alpha\}$ is tradable with some bid in B . Therefore, α is the most-competitive ask in $A \cup \{\alpha\}$ and all transactions $t \in M$ are such that $\text{id}_{\text{ask}}(t) = \text{id}(\alpha)$. Thus no a' exists such that $a \succ a'$ and $\text{id}(a') \in \text{ids}_{\text{ask}}(M)$. Thus, price-time priority trivially holds.

Next, we prove (ii) by induction on $|B|$.

Base case is trivial as $B = \emptyset \implies M = \emptyset$.

Induction hypothesis: We assume that (ii) holds for all B such that $|B| < k$.

Induction step: We now show that (ii) holds for all B such that $|B| = k$.

Fix $b, b' \in B$ such that $b \succ b'$ and b' gets traded in M . We need to show that b is fully traded in M .

Let β be the most competitive bid in B . If β is not tradable with α , as per `Match_Ask`, $M = \emptyset$ and we are trivially done. Henceforth we assume β is tradable with α .

Observe, `Match_Ask` is such that it completely trades β in its output M (along with possibly other bids from B) or only trades a part of β but no other bid from B . Thus, β cannot be b' . We thus consider two cases: $b = \beta$ or $b \neq \beta$.

- Case: $b = \beta$. Here again, we are done from the above observation that either β gets completely traded or β gets partially traded but no other bid gets traded at all.
- Case: $b \neq \beta$. When $\text{qty}(\beta) = \text{qty}(\alpha)$ or $\text{qty}(\beta) > \text{qty}(\alpha)$, as per `Match_Ask`, β is the only bid that gets partially or fully traded in M , contradicting the existence of b and b' .

In the final case, when $\text{qty}(\beta) < \text{qty}(\alpha)$, `Match_Ask` on (B, A, α) recursively calls `Match_Ask` on $(B \setminus \{\beta\}, A, \alpha')$ where α' is obtained from α by reducing its quantity

by $\text{qty}(\beta)$. Let the recursive call return $(B', A', M') = \text{Match_Ask}(B \setminus \{\beta\}, A, \alpha')$, then Match_Ask returns $(\hat{B}, \hat{A}, M) = (B', A', M \cup \{m\})$, where m is a transaction between α and β . Thus, all transactions t in M where $\text{id}_{\text{bid}}(t) \neq \text{id}(\beta)$ come from M' . Now since $\beta \succ b \succ b'$ are distinct, if b' gets traded in M , it must get traded in M' and applying the induction hypothesis (as it is easy to verify that $|B \setminus \{\beta\}| = k - 1$, $(B \setminus \{\beta\}, A \cup \{\alpha'\})$ admissible and $(B \setminus \{\beta\}, A)$ not matchable), we have b gets fully traded in M' and hence in M (as $M = M' \cup \{m\}$) and we are done. \blacktriangleleft

A.3 Match_Ask satisfies conservation

Match_Ask satisfies conservation follows immediately from the following lemma.

► **Lemma 3.** *Let $(B, A \cup \{\alpha\})$ be an admissible order-domain. If $\text{Match_Ask}(B, A, \alpha) = (\hat{B}, \hat{A}, M)$, then*

- a. M is matching over the order-domain $(B, A \cup \{\alpha\})$
- b. $\hat{B} = B - \text{Bids}(M, B)$
- c. $\hat{A} = (A \cup \{\alpha\}) - \text{Asks}(M, A \cup \{\alpha\})$.

Proof. We briefly outline the proof stressing only the important aspects.

Part a. To show M is a matching over $(B, A \cup \{\alpha\})$, we need to show that (i) each transaction in M is between a tradable bid-ask pair where the bid is in B and the ask is in $A \cup \{\alpha\}$, and (ii) for each order ω in $B \cup A \cup \{\alpha\}$, its total transaction quantity in M $\text{Qty}(M, \text{id}(\omega))$ is at most its quantity $\text{qty}(\omega)$.

We will show this by induction on $|B|$. The base case is trivial: $|B| = 0 \implies M = \emptyset$, which is trivially a matching. We assume that (i) and (ii) hold for all B such that $|B| < k$. We now show that (i) and (ii) hold assuming $|B| = k$.

Let β be the most competitive bid in B . When β is not tradable with α , as per Match_Ask , $M = \emptyset$ and we are trivially done. Henceforth we assume β is tradable with α . Now, when $\text{qty}(\beta) \geq \text{qty}(\alpha)$, Match_Ask outputs only a single transaction m between β and α with transaction quantity $\text{qty}(m) = \text{qty}(\alpha) \leq \text{qty}(\beta)$ and clearly $M = \{m\}$ satisfies (i) and (ii) above.

When β is tradable with α and $\text{qty}(\beta) < \text{qty}(\alpha)$, Match_Ask creates a transaction m between β and α with transaction quantity $\text{qty}(\beta)$ and obtains α' from α by reducing its quantity by $\text{qty}(\beta)$. It then recursively calls $\text{Match_Ask}(B \setminus \{\beta\}, A, \alpha')$ to obtain the matching M' and then outputs the matching $M = M' \cup \{m\}$.

Note that $(B, A \cup \{\alpha\})$ admissible implies $(B \setminus \{\beta\}, A \cup \{\alpha'\})$ is admissible. Furthermore, since $|B \setminus \{\beta\}| = k - 1$, we can apply the induction hypothesis to obtain that M' is a matching between $B \setminus \{\beta\}$ and $A \cup \{\alpha'\}$. In particular, this means that β does not participate in M' and the total transaction quantity of α in M' is at most $\text{qty}(\alpha') = \text{qty}(\alpha) - \text{qty}(\beta)$. So the total transaction quantity of β in $M = M' \cup \{m\}$ is the transaction quantity of m , which is $\text{qty}(\beta)$, and the total transaction quantity of α in $M = M' \cup \{m\}$ is at most $\text{qty}(\alpha') + \text{qty}(m) = (\text{qty}(\alpha) - \text{qty}(\beta)) + \text{qty}(\beta) = \text{qty}(\alpha)$. With these facts, one can easily verify that (i) and (ii) hold.

Part b. Here we need to show that $\hat{B} = B - \text{Bids}(M, B)$. Once again we argue by induction on $|B|$. Let β be the most competitive bid in B .

In all but the last if statement of `Match_Ask` (where β and α are tradable and $\text{qty}(\beta) < \text{qty}(\alpha)$), we have B , \hat{B} , and M explicitly and we can directly verify that the property holds.

In the last if condition, `Match_Ask` first creates a transaction m between β and α with transaction quantity $\text{qty}(\beta)$ and obtains α' from α by reducing its quantity to $\text{qty}(\alpha) - \text{qty}(\beta)$. It then computes $(B', A', M') = \text{Match_Ask}(B \setminus \{\beta\}, A, \alpha')$ recursively. Finally it returns $(\hat{B}, \hat{A}, M) = (B', A', M' \cup \{m\})$.

For the recursive call, we can apply the induction hypothesis. Thus,

$$\begin{aligned} B' &= (B \setminus \{\beta\}) - \text{Bids}(M', B \setminus \{\beta\}) && \text{(from I.H.)} \\ \implies \hat{B} &= (B \setminus \{\beta\}) - \text{Bids}(M', B \setminus \{\beta\}) && \text{(since } \hat{B} = B') \\ &= B - \text{Bids}(M, B), \end{aligned}$$

where the last equality follows from the observation that β is missing from both $B \setminus \{\beta\}$ and $\text{Bids}(M', B \setminus \{\beta\})$, and $\text{Bids}(M, B)$ contains β (as $M = M' \cup \{m\}$ where m is a transaction involving β with transaction quantity $\text{qty}(\beta)$).

Part c. Observe that all transactions produced by `Match_Ask` have α as its participating ask. Thus, one can check that we will be done if we can show that

$$\text{qty}(\hat{A}, \text{id}(\alpha)) = \text{qty}(\alpha) - \text{Qty}(M, \text{id}(\alpha)).$$

To show the above, we again induct on $|B|$. In all but the last if statement of `Match_Ask`, we have α , \hat{A} , and M explicitly, and we can directly verify that the above equation holds.

In the last if statement, we have that $\text{qty}(\beta) < \text{qty}(\alpha)$ and β and α are tradable, where β is the most competitive bid in B . In this case, `Match_Ask` creates a transaction m between β and α with transaction quantity $\text{qty}(\beta)$ and obtains α' from α by reducing its quantity by $\text{qty}(\beta)$. Then, it recursively calls `Match_Ask` on $(B \setminus \{\beta\}, A, \alpha')$ to obtain (B', A', M') and finally outputs $(\hat{B}, \hat{A}, M) = (B', A', M' \cup \{m\})$. We can apply the induction hypothesis for the recursive call. Thus,

$$\begin{aligned} \text{qty}(A', \text{id}(\alpha')) &= \text{qty}(\alpha') - \text{Qty}(M', \text{id}(\alpha')) && \text{(from I.H.)} \\ \implies \text{qty}(\hat{A}, \text{id}(\alpha)) &= \text{qty}(\alpha') - \text{Qty}(M', \text{id}(\alpha)) \\ &\quad \text{(since } \hat{A} = A' \text{ and } \text{id}(\alpha') = \text{id}(\alpha)) \\ &= \text{qty}(\alpha) - [\text{qty}(\beta) + \text{Qty}(M', \text{id}(\alpha))] \\ &\quad \text{(as } \text{qty}(\alpha') = \text{qty}(\alpha) - \text{qty}(\beta)) \\ &= \text{qty}(\alpha) - \text{Qty}(M, \text{id}(\alpha)), \end{aligned}$$

where the last equality follows from observing that $M = M' \cup \{m\}$ and m is a transaction involving α with transaction quantity $\text{qty}(\beta)$. ◀

Parikh Automata over Infinite Words

Shibashis Guha  

Tata Institute of Fundamental Research, Mumbai, India

Ismaël Jecker  

University of Warsaw, Poland

Karoliina Lehtinen  

CNRS, Aix-Marseille University, LIS, Marseille, France

Martin Zimmermann  

Aalborg University, Denmark

Abstract

Parikh automata extend finite automata by counters that can be tested for membership in a semilinear set, but only at the end of a run, thereby preserving many of the desirable algorithmic properties of finite automata. Here, we study the extension of the classical framework onto infinite inputs: We introduce reachability, safety, Büchi, and co-Büchi Parikh automata on infinite words and study expressiveness, closure properties, and the complexity of verification problems.

We show that almost all classes of automata have pairwise incomparable expressiveness, both in the deterministic and the nondeterministic case; a result that sharply contrasts with the well-known hierarchy in the ω -regular setting. Furthermore, emptiness is shown decidable for Parikh automata with reachability or Büchi acceptance, but undecidable for safety and co-Büchi acceptance. Most importantly, we show decidability of model checking with specifications given by deterministic Parikh automata with safety or co-Büchi acceptance, but also undecidability for all other types of automata. Finally, solving games is undecidable for all types.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory

Keywords and phrases Parikh automata, ω -automata, Infinite Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.40

Related Version *Full Version:* <https://arxiv.org/abs/2207.07694> [17]

Funding *Shibashis Guha:* Supported by the DST-SERB project SRG/2021/000466.

Ismaël Jecker: Supported by the ERC grant 950398 (INFSYS).

Martin Zimmermann: Supported by DIREC – Digital Research Centre Denmark.

Acknowledgements We want to thank an anonymous reviewer for proposing Lemma 14.

1 Introduction

While finite-state automata are the keystone of automata-theoretic verification, they are not expressive enough to deal with the many nonregular aspects of realistic verification problems. Various extensions of finite automata have emerged over the years, to allow for the specification of context-free properties and beyond, as well as the modelling of timed and quantitative aspects of systems. Among these extensions, Parikh automata, introduced by Klaedtke and Rueß [19], consist of finite automata augmented with counters that can only be incremented. A Parikh automaton only accepts a word if the final counter-configuration is within a semilinear set specified by the automaton. As the counters do not interfere with the control flow of the automaton, that is, counter values do not affect whether transitions are enabled, they allow for mild quantitative computations without the full power of vector addition systems or other more powerful models.



© Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann;
licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 40; pp. 40:1–40:20



Leibniz International Proceedings in Informatics

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

For example, the nonregular language of words that have more a 's than b 's is accepted by a Parikh automaton obtained from the one-state DFA accepting $\{a, b\}^*$ by equipping it with two counters, one counting the a 's in the input, the other counting the b 's, and a semilinear set ensuring that the first counter is larger than the second one. With a similar approach, one can construct a Parikh automaton accepting the non-context-free language of words that have more a 's than b 's and more a 's than c 's.

Klaedtke and Rueß [19] showed Parikh automata to be expressively equivalent to a quantitative version of existential WMSO that allows for reasoning about set cardinalities. Their expressiveness also coincides with that of reversal-bounded counter machines [19], in which counters can go from decrementing to incrementing only a bounded number of times, but in which counters affect control flow [18]. The (weakly) unambiguous restriction of Parikh automata, that is, those that have at most one accepting run, on the other hand, coincide with unambiguous reversal-bounded counter machines [2]. Parikh automata are also expressively equivalent to weighted finite automata over the groups $(\mathbb{Z}^k, +, 0)$ [9, 21] for $k \geq 1$. This shows that Parikh automata accept a natural class of quantitative specifications.

Despite their expressiveness, Parikh automata retain some decidability: nonemptiness, in particular, is NP-complete [12]. For weakly unambiguous Parikh automata, inclusion [5] and regular separability [6] are decidable as well. Figueira and Libkin [12] also argued that this model is well-suited for querying graph databases, while mitigating some of the complexity issues related with more expressive query languages. Further, they have been used in the model checking of transducer properties [14].

As Parikh automata have been established as a robust and useful model, many variants thereof exist: pushdown (visibly [8] and otherwise [23]), two-way with [8] and without stack [13], unambiguous [4], and weakly unambiguous [2] Parikh automata, to name a few. Despite this attention, so far, some more elementary questions have remained unanswered. For instance, despite Klaedtke and Rueß's suggestion in [19] that the model could be extended to infinite words, we are not aware of previous work on ω -Parikh automata.

Yet, specifications over infinite words are a crucial part of the modern verification landscape. Indeed, programs, especially safety-critical ones, are often expected to run continuously, possibly in interaction with an environment. Then, executions are better described by infinite words, and accordingly, automata over infinite, rather than finite, words are appropriate for capturing specifications.

This is the starting point of our contribution: we extend Parikh automata to infinite inputs, and consider reachability, safety, Büchi, and co-Büchi acceptance conditions. We observe that when it comes to reachability and Büchi, there are two possible definitions: an asynchronous one that just requires both an accepting state and the semilinear set to be reached (once or infinitely often) by the run, but not necessarily at the same time, and a synchronous one that requires both to be reached (once or infinitely often) simultaneously. Parikh automata on infinite words accept, for example, the languages of infinite words

- with some prefix having more a 's than b 's (reachability acceptance),
- with all nonempty prefixes having more a 's than b 's (safety acceptance),
- with infinitely many prefixes having more a 's than b 's (Büchi acceptance), and
- with almost all prefixes having more a 's than b 's (co-Büchi acceptance).

We establish that, both for reachability and Büchi acceptance, both the synchronous and the asynchronous variant are linearly equivalent in the presence of nondeterminism, but not for deterministic automata. Hence, by considering all acceptance conditions and (non)determinism, we end up with twelve different classes of automata. We show that almost all of these classes have pairwise incomparable expressiveness, which is in sharp contrast

to the well-known hierarchies in the ω -regular case. Furthermore, we establish an almost complete picture of the Boolean closure properties of these twelve classes of automata. Most notably, they lack closure under negation, even for nondeterministic Büchi Parikh automata. Again, this result should be contrasted with the ω -regular case, where nondeterministic Büchi automata are closed under negation [22].

We then study the complexity of the most important verification problems, e.g., non-emptiness, universality, model checking, and solving games. We show that nonemptiness is undecidable for deterministic safety and co-Büchi Parikh automata. However, perhaps surprisingly, we also show that nonemptiness is decidable, in fact NP-complete, for reachability and Büchi Parikh automata, both for the synchronous and the asynchronous versions. Strikingly, for Parikh automata, the Büchi acceptance condition is algorithmically simpler than the safety one (recall that their expressiveness is pairwise incomparable).

Next, we consider model checking, arguably the most successful application of automata theory in the field of automated verification. Model checking asks whether a given finite-state system satisfies a given specification. Here, we consider quantitative specifications given by Parikh automata. Model checking is decidable for specifications given by deterministic Parikh automata with safety or co-Büchi acceptance. On the other hand, the problem is undecidable for all other classes of automata.

The positive results imply that one can model-check an arbiter serving requests from two clients against specifications like “the accumulated waiting time between requests and responses of client 1 is always at most twice the accumulated waiting time for client 2 and vice versa” and “the difference between the number of responses for client 1 and the number of responses for client 2 is from some point onward bounded by 100”. Note that both properties are not ω -regular.

Finally, we consider solving games with winning conditions expressed by Parikh automata. Zero-sum two-player games are a key formalism used to model the interaction of programs with an uncontrollable environment. In particular, they are at the heart of solving synthesis problems in which, rather than verifying the correctness of an existing program, we are interested in generating a program that is correct by construction, from its specifications. In these games, the specification corresponds to the winning condition: one player tries to build a word (i.e., behaviour) that is in the specification, while the other tries to prevent this. As with model checking, using Parikh automata to capture the specification would enable these well-understood game-based techniques to be extended to mildly quantitative specifications. However, we show that games with winning conditions specified by Parikh automata are undecidable for all acceptance conditions we consider.

All proofs omitted due to space restrictions can be found in the full version [17].

2 Definitions

An alphabet is a finite nonempty set Σ of letters. As usual, ε denotes the empty word, Σ^* (Σ^+ , Σ^ω) denotes the set of finite (finite nonempty, infinite) words over Σ . The length of a finite word w is denoted by $|w|$ and, for notational convenience, we define $|w| = \infty$ for infinite words w .

The number of occurrences of the letter a in a finite word w is denoted by $|w|_a$. Let $a, b \in \Sigma$. A word $w \in \Sigma^*$ is (a, b) -balanced if $|w|_a = |w|_b$, otherwise it is (a, b) -unbalanced. Note that the empty word is (a, b) -balanced.

Semilinear Sets. Let \mathbb{N} denote the set of nonnegative integers. Let $\vec{v} = (v_0, \dots, v_{d-1}) \in \mathbb{N}^d$ and $\vec{v}' = (v'_0, \dots, v'_{d'-1}) \in \mathbb{N}^{d'}$ be a pair of vectors. We define their concatenation as $\vec{v} \cdot \vec{v}' = (v_0, \dots, v_{d-1}, v'_0, \dots, v'_{d'-1}) \in \mathbb{N}^{d+d'}$. We lift the concatenation of vectors to sets $D \subseteq \mathbb{N}^d$ and $D' \subseteq \mathbb{N}^{d'}$ via $D \cdot D' = \{\vec{v} \cdot \vec{v}' \mid \vec{v} \in D \text{ and } \vec{v}' \in D'\}$.

Let $d \geq 1$. A set $C \subseteq \mathbb{N}^d$ is *linear* if there are vectors $\vec{v}_0, \dots, \vec{v}_k \in \mathbb{N}^d$ such that

$$C = \left\{ \vec{v}_0 + \sum_{i=1}^k c_i \vec{v}_i \mid c_i \in \mathbb{N} \text{ for } i = 1, \dots, k \right\}.$$

Furthermore, a subset of \mathbb{N}^d is *semilinear* if it is a finite union of linear sets.

► **Proposition 1** ([16]). *If $C, C' \subseteq \mathbb{N}^d$ are semilinear, then so are $C \cup C'$, $C \cap C'$, $\mathbb{N}^d \setminus C$, as well as $\mathbb{N}^{d'} \cdot C$ and $C \cdot \mathbb{N}^{d'}$ for every $d' \geq 1$.*

Finite Automata. A (nondeterministic) finite automaton (NFA) $\mathcal{A} = (Q, \Sigma, q_I, \Delta, F)$ over Σ consists of a finite set Q of states containing the initial state q_I , an alphabet Σ , a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and a set $F \subseteq Q$ of accepting states. The NFA is deterministic (i.e., a DFA) if for every state $q \in Q$ and every letter $a \in \Sigma$, there is at most one $q' \in Q$ such that (q, a, q') is a transition of \mathcal{A} . A run of \mathcal{A} is a (possibly empty) sequence $(q_0, w_0, q_1)(q_1, w_1, q_2) \cdots (q_{n-1}, w_{n-1}, q_n)$ of transitions with $q_0 = q_I$. It processes the word $w_0 w_1 \cdots w_{n-1} \in \Sigma^*$. The run is *accepting* if it is either empty and the initial state is accepting or if it is nonempty and q_n is accepting. The language $L(\mathcal{A})$ of \mathcal{A} contains all finite words $w \in \Sigma^*$ such that \mathcal{A} has an accepting run processing w .

Parikh Automata. Let Σ be an alphabet, $d \geq 1$, and D a finite subset of \mathbb{N}^d . Furthermore, let $w = (a_0, \vec{v}_0) \cdots (a_{n-1}, \vec{v}_{n-1})$ be a word over $\Sigma \times D$. The Σ -projection of w is $p_\Sigma(w) = a_0 \cdots a_{n-1} \in \Sigma^*$ and its *extended Parikh image* is $\Phi_e(w) = \sum_{j=0}^{n-1} \vec{v}_j \in \mathbb{N}^d$ with the convention $\Phi_e(\varepsilon) = \vec{0}$, where $\vec{0}$ is the d -dimensional zero vector.

A *Parikh automaton* (PA) is a pair (\mathcal{A}, C) such that \mathcal{A} is an NFA over $\Sigma \times D$ for some input alphabet Σ and some finite $D \subseteq \mathbb{N}^d$ for some $d \geq 1$, and $C \subseteq \mathbb{N}^d$ is semilinear. The language of (\mathcal{A}, C) consists of the Σ -projections of words $w \in L(\mathcal{A})$ whose extended Parikh image is in C , i.e.,

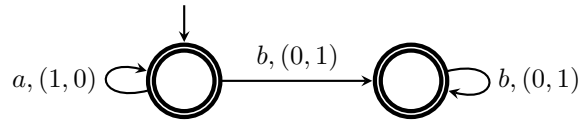
$$L(\mathcal{A}, C) = \{p_\Sigma(w) \mid w \in L(\mathcal{A}) \text{ with } \Phi_e(w) \in C\}.$$

The automaton (\mathcal{A}, C) is deterministic, if for every state q of \mathcal{A} and every $a \in \Sigma$, there is at most one pair $(q', \vec{v}) \in Q \times D$ such that $(q, (a, \vec{v}), q')$ is a transition of \mathcal{A} . Note that this definition does *not* coincide with \mathcal{A} being deterministic: As mentioned above, \mathcal{A} accepts words over $\Sigma \times D$ while (\mathcal{A}, C) accepts words over Σ . Therefore, determinism is defined with respect to Σ only.

Note that the above definition of $L(\mathcal{A}, C)$ coincides with the following alternative definition via accepting runs: A run ρ of (\mathcal{A}, C) is a run

$$\rho = (q_0, (a_0, \vec{v}_0), q_1)(q_1, (a_1, \vec{v}_1), q_2) \cdots (q_{n-1}, (a_{n-1}, \vec{v}_{n-1}), q_n)$$

of \mathcal{A} . We say that ρ *processes* the word $a_0 a_1 \cdots a_{n-1} \in \Sigma^*$, i.e., the \vec{v}_j are ignored, and that ρ 's extended Parikh image is $\sum_{j=0}^{n-1} \vec{v}_j$. The run is *accepting*, if it is either empty and both the initial state of \mathcal{A} is accepting and the zero vector (the extended Parikh image of the empty run) is in C , or if it is nonempty, q_n is accepting, and ρ 's extended Parikh image is in C . Finally, (\mathcal{A}, C) accepts $w \in \Sigma^*$ if it has an accepting run processing w .



■ **Figure 1** The automaton for Example 2.

► **Example 2.** Consider the deterministic PA (\mathcal{A}, C) with \mathcal{A} in Figure 1 and $C = \{(n, n) \mid n \in \mathbb{N}\} \cup \{(n, 2n) \mid n \in \mathbb{N}\}$. It accepts the language $\{a^n b^n \mid n \in \mathbb{N}\} \cup \{a^n b^{2n} \mid n \in \mathbb{N}\}$.

A cycle is a nonempty finite run infix

$$(q_0, w_0, q_1)(q_1, w_1, q_2) \cdots (q_{n-1}, w_{n-1}, q_n)(q_n, w_n, q_0)$$

starting and ending in the same state and such that the q_j are pairwise different. Note that every run infix containing at least n transitions contains a cycle, where n is the number of states of the automaton. Many of our proofs rely on the following shifting argument, which has been used before to establish inexpressibility results for Parikh automata [3].

► **Remark 3.** Let $\rho_0 \rho_1 \rho_2 \rho_3$ be a run of a PA such that ρ_1 and ρ_3 are cycles starting in the same state. Then, $\Phi_e(\rho_0 \rho_1 \rho_2 \rho_3) = \Phi_e(\rho_0 \rho_2 \rho_1 \rho_3) = \Phi_e(\rho_0 \rho_1 \rho_3 \rho_2)$. Furthermore, all three runs end in the same state and visit the same set of states (but maybe in different orders).

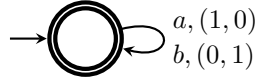
3 Parikh Automata over Infinite Words

In this section, we introduce Parikh automata over infinite words by lifting safety, reachability, Büchi, and co-Büchi acceptance from finite automata to Parikh automata. Recall that a Parikh automaton on finite words accepts if the last state of the run is accepting and the extended Parikh image of the run is in the semilinear set, i.e., both events are synchronized. For reachability and Büchi acceptance it is natural to consider both a synchronous and an asynchronous variant while for safety and co-Büchi there is only a synchronous variant.

All these automata have the same format as Parikh automata on finite words, but are now processing infinite words. Formally, consider (\mathcal{A}, C) with $\mathcal{A} = (Q, \Sigma \times D, q_I, \Delta, F)$. Fix an infinite run $(q_0, w_0, q_1)(q_1, w_1, q_2)(q_2, w_2, q_3) \cdots$ of \mathcal{A} with $q_0 = q_I$ (recall that each w_j is in $\Sigma \times D$), which we say *processes* $p_\Sigma(w_0 w_1 w_2 \cdots)$.

- The run is *safety accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for all $n \geq 0$.
- The run is *synchronous reachability accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for some $n \geq 0$.
- The run is *asynchronous reachability accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ for some $n \geq 0$ and $q_{n'} \in F$ for some $n' \geq 0$.
- The run is *synchronous Büchi accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for infinitely many $n \geq 0$.
- The run is *asynchronous Büchi accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ for infinitely many $n \geq 0$ and $q_{n'} \in F$ for infinitely many $n' \geq 0$.
- The run is *co-Büchi accepting* if there is an n_0 such that $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for every $n \geq n_0$.

As mentioned before, we do not distinguish between synchronous and asynchronous co-Büchi acceptance, as these definitions are equivalent. Also, note that all our definitions are conjunctive in the sense that acceptance requires visits to accepting states *and* extended Parikh images in C . Thus, e.g., reachability and safety are not dual on a syntactic level. Nevertheless, we later prove dualities on a semantic level.



■ **Figure 2** The automaton for Example 4.

Similarly, one can easily show that a disjunctive definition is equivalent to our conjunctive one: One can reflect in the extended Parikh image of a run prefix whether it ends in an accepting state and then encode acceptance in the semilinear set. So, any given Parikh automaton (\mathcal{A}, C) (with disjunctive or conjunctive acceptance) can be turned into another one (\mathcal{A}', C') capturing acceptance in (\mathcal{A}, C) by Parikh images only. So, with empty (full) set of accepting states and C' mimicking disjunction (conjunction), it is equivalent to the original automaton with disjunctive (conjunctive) acceptance.

Now, the language $L_S(\mathcal{A}, C)$ of a safety Parikh automaton (SPA) (\mathcal{A}, C) contains those words $w \in \Sigma^\omega$ such that (\mathcal{A}, C) has a safety accepting run processing w . Similarly, we define the languages

- $L_R^s(\mathcal{A}, C)$ of synchronous reachability Parikh automata (sRPA),
- $L_R^a(\mathcal{A}, C)$ of asynchronous reachability Parikh automata (aRPA),
- $L_B^s(\mathcal{A}, C)$ of synchronous Büchi Parikh automata (aBPA),
- $L_B^a(\mathcal{A}, C)$ of asynchronous Büchi Parikh automata (sBPA), and
- $L_C(\mathcal{A}, C)$ of co-Büchi Parikh automata (CPA).

Determinism for all types of automata is defined as for Parikh automata on finite words. Unless explicitly stated otherwise, every automaton is assumed to be nondeterministic.

► **Example 4.** Let \mathcal{A} be the DFA shown in Figure 2 and let $C = \{(n, n) \mid n \in \mathbb{N}\}$ and $\overline{C} = \{(n, n') \mid n \neq n'\} = \mathbb{N}^2 \setminus C$.

Recall that a finite word w is (a, b) -balanced if $|w|_a = |w|_b$, i.e., the number of a 's and b 's in w is equal. The empty word is (a, b) -balanced and every odd-length word over $\{a, b\}$ is (a, b) -unbalanced.

1. When interpreting (\mathcal{A}, C) as a PA, it accepts the language of finite (a, b) -balanced words; when interpreting $(\mathcal{A}, \overline{C})$ as a PA, it accepts the language of finite (a, b) -unbalanced words.
2. When interpreting (\mathcal{A}, C) as an aRPA or sRPA, it accepts the language of infinite words that have an (a, b) -balanced prefix; when interpreting $(\mathcal{A}, \overline{C})$ as an aRPA or sRPA, it accepts the language of infinite words that have an (a, b) -unbalanced prefix. Note that both languages are universal, as the empty prefix is always (a, b) -balanced and every odd-length prefix is (a, b) -unbalanced.
3. When interpreting (\mathcal{A}, C) as an SPA, it accepts the language of infinite words that have only (a, b) -balanced prefixes; when interpreting $(\mathcal{A}, \overline{C})$ as an SPA, it accepts the language of infinite words that have only (a, b) -unbalanced prefixes. Here, both languages are empty, which follows from the same arguments as for universality in the previous case.
4. When interpreting (\mathcal{A}, C) as an aBPA or sBPA, it accepts the language of infinite words with infinitely many (a, b) -balanced prefixes; when interpreting $(\mathcal{A}, \overline{C})$ as an aBPA or sBPA, it accepts the language of infinite words with infinitely many (a, b) -unbalanced prefixes. The latter language is universal, as every odd-length prefix is unbalanced.
5. When interpreting (\mathcal{A}, C) as a CPA, it accepts the language of infinite words such that almost all prefixes are (a, b) -balanced; when interpreting $(\mathcal{A}, \overline{C})$ as a CPA, it accepts the language of infinite words such that almost all prefixes are (a, b) -unbalanced. Again, the former language is empty.

Let (\mathcal{A}, C) be a Parikh automaton. We say that a run prefix is an F -prefix if it ends in an accepting state of \mathcal{A} , a C -prefix if its extended Parikh image is in C , and an FC -prefix if it is both an F -prefix and a C -prefix. Note that both asynchronous acceptance conditions are defined in terms of the existence of F -prefixes and C -prefixes and all other acceptance conditions in terms of the existence of FC -prefixes.

► **Remark 5.** sRPA and aRPA (SPA, sBPA and aBPA, CPA) are strictly more expressive than ω -regular reachability (safety, Büchi, co-Büchi) automata. Inclusion follows by definition while strictness is witnessed by the languages presented in Example 4.

4 Expressiveness

In this section, we study the expressiveness of the various types of Parikh automata on infinite words introduced above, by comparing synchronous and asynchronous variants, deterministic and nondeterministic variants, and the different acceptance conditions.

► **Remark 6.** In this, and only this, section, we consider only reachability Parikh automata that are complete in the following sense: For every state q and every letter a there is a vector \vec{v} and a state q' such that $(q, (a, \vec{v}), q')$ is a transition of \mathcal{A} , i.e., every letter can be processed from every state. Without this requirement, one can express safety conditions by incompleteness, while we want to study the expressiveness of “pure” reachability automata.

Safety, Büchi, and co-Büchi automata can be assumed, without loss of generality, to be complete, as one can always add a nonaccepting sink to complete such an automaton without modifying the accepted language.

We begin our study by comparing the synchronous and asynchronous variants of reachability and Büchi automata. All transformations proving the following inclusions are effective and lead to a linear increase in the number of states and a constant increase in the dimension of the semilinear sets.

► Theorem 7.

1. *aRPA and sRPA are equally expressive.*
2. *Deterministic aRPA are strictly more expressive than deterministic sRPA.*
3. *aBPA and sBPA are equally expressive.*
4. *Deterministic aBPA are strictly more expressive than deterministic sBPA.*

Due to the equivalence of synchronous and asynchronous (nondeterministic) reachability Parikh automata, we drop the qualifiers whenever possible and just speak of reachability Parikh automata (RPA). We do the same for (nondeterministic) Büchi Parikh automata (BPA).

Next, we compare the deterministic and nondeterministic variants for each acceptance condition. Note that all separations are as strong as possible, i.e., for reachability and Büchi we consider deterministic asynchronous automata, which are more expressive than their synchronous counterparts (see Theorem 7).

► Theorem 8.

1. *Nondeterministic RPA are strictly more expressive than deterministic aRPA.*
2. *Nondeterministic SPA are strictly more expressive than deterministic SPA.*
3. *Nondeterministic BPA are strictly more expressive than deterministic aBPA.*
4. *Nondeterministic CPA are strictly more expressive than deterministic CPA.*

After having separated deterministic and nondeterministic automata for all acceptance conditions, we now consider inclusions and separations between the different acceptance conditions. Here, the picture is notably different than in the classical ω -regular setting, as almost all classes can be separated.

► **Theorem 9.** *Every RPA can be turned into an equivalent BPA and into an equivalent CPA. All other automata types are pairwise incomparable.*

Our separations between the different acceptance conditions are as strong as possible, e.g., when we show that not every RPA has an equivalent SPA, we exhibit a *deterministic sRPA* (the weakest class of RPA) whose language is not accepted by any nondeterministic SPA (the strongest class of SPA). The same is true for all other separations.

5 Closure Properties

In this section, we study the closure properties of Parikh automata on infinite words. We begin by showing that, for deterministic synchronous automata, reachability and safety acceptance as well as Büchi and co-Büchi acceptance are dual, although they are not syntactically dual due to all acceptance conditions being defined by a conjunction. On the other hand, deterministic asynchronous automata can still be complemented, however only into nondeterministic automata.

► **Theorem 10.**

1. Let (\mathcal{A}, C) be a deterministic sRPA. The complement of $L_R^s(\mathcal{A}, C)$ is accepted by a deterministic SPA.
2. Let (\mathcal{A}, C) be a deterministic aRPA. The complement of $L_R^a(\mathcal{A}, C)$ is accepted by an SPA, but not necessarily by a deterministic SPA.
3. Let (\mathcal{A}, C) be a deterministic SPA. The complement of $L_S(\mathcal{A}, C)$ is accepted by a deterministic sRPA.
4. Let (\mathcal{A}, C) be a deterministic sBPA. The complement of $L_B^s(\mathcal{A}, C)$ is accepted by a deterministic CPA.
5. Let (\mathcal{A}, C) be a deterministic aBPA. The complement of $L_B^a(\mathcal{A}, C)$ is accepted by a CPA, but not necessarily by a deterministic CPA.
6. Let (\mathcal{A}, C) be a deterministic CPA. The complement of $L_C(\mathcal{A}, C)$ is accepted by a deterministic sBPA.

The positive results above are for deterministic automata. For nondeterministic automata, the analogous statements fail.

► **Theorem 11.**

1. There exists an sRPA (\mathcal{A}, C) such that no SPA accepts the complement of $L_R^s(\mathcal{A}, C)$.
2. There exists an SPA (\mathcal{A}, C) such that no RPA accepts the complement of $L_S(\mathcal{A}, C)$.
3. There exists an sBPA (\mathcal{A}, C) such that no CPA accepts the complement of $L_B^s(\mathcal{A}, C)$.
4. There exists a CPA (\mathcal{A}, C) such that no BPA accepts the complement of $L_C(\mathcal{A}, C)$.

Next, we consider closure under union, intersection, and complementation of the various classes of Parikh automata on infinite words. Notably, all nondeterministic (and some deterministic) classes are closed under union, the picture for intersection is more scattered, and we prove failure of complement closure for all classes. Again, this is in sharp contrast to the setting of classical Büchi automata, which are closed under all three Boolean operations.

► **Theorem 12.** *The closure properties depicted in Table 1 hold.*

■ **Table 1** Closure properties and decidability of decision problems for Parikh automata on infinite words.

	Closure			Decision Problems			
	\cup	\cap	$-$	Emptiness	Universality	Model Check.	Games
RPA	✓	✓	✗	NP-compl.	undec.	undec.	undec.
det. aRPA	✗	✗	✗	NP-compl.	undec.	undec.	undec.
det. sRPA	✓	✗	✗	NP-compl.	undec.	undec.	undec.
SPA	✓	✓	✗	undec.	undec.	undec.	undec.
det. SPA	✗	✓	✗	undec.	coNP-compl.	coNP-compl.	undec.
BPA	✓	✗	✗	NP-compl.	undec.	undec.	undec.
det. aBPA	?	✗	✗	NP-compl.	undec.	undec.	undec.
det. sBPA	✓	✗	✗	NP-compl.	undec.	undec.	undec.
CPA	✓	✓	✗	undec.	undec.	undec.	undec.
det. CPA	✗	✓	✗	undec.	coNP-compl.	coNP-compl.	undec.

Note that there is one question mark in the closure properties columns in Table 1, which we leave for further research.

6 Decision Problems

In this section, we study the complexity of the nonemptiness and the universality problem, model checking, and solving games for Parikh automata on infinite words. Before we can do so, we need to specify how a Parikh automaton (\mathcal{A}, C) is represented as input for algorithms: The vectors labeling the transitions of \mathcal{A} are represented in binary and a linear set

$$\left\{ \vec{v}_0 + \sum_{i=1}^k c_i \vec{v}_i \mid c_i \in \mathbb{N} \text{ for } i = 1, \dots, k \right\}$$

is represented by the list $(\vec{v}_0, \dots, \vec{v}_k)$ of vectors, again encoded in binary. A semilinear set is then represented by a set of such lists.

6.1 Nonemptiness

We begin by settling the complexity of the nonemptiness problem. The positive results are obtained by reductions to the emptiness of Parikh automata on finite words while the undecidability results are reductions from the termination problem for two-counter machines.

► **Theorem 13.** *The following problems are NP-complete:*

1. *Given an RPA, is its language nonempty?*
2. *Given a BPA, is its language nonempty?*

The following problems are undecidable:

3. *Given a deterministic SPA, is its language nonempty?*
4. *Given a deterministic CPA, is its language nonempty?*

Proof.

1. Due to Theorem 7.1, we only consider the case of sRPA for the NP upper bound. Given such an automaton (\mathcal{A}, C) with $\mathcal{A} = (Q, \Sigma \times D, q_I, \Delta, F)$ let $F' \subseteq F$ be the set of accepting states from which a cycle is reachable. Now, define $\mathcal{A}' = (Q, \Sigma \times D, q_I, \Delta, F')$. Then, we have $L_R^s(\mathcal{A}, C) \neq \emptyset$ if and only if $L(\mathcal{A}', C) \neq \emptyset$ (i.e., we treat (\mathcal{A}', C) as a PA), with the latter problem being in NP [12].

The matching NP lower bound is, again due to Theorem 7.1, only shown for sRPA. We proceed by a reduction from the NP-complete [12] nonemptiness problem for Parikh automata. Given a Parikh automaton (\mathcal{A}, C) , let \mathcal{A}' be obtained from \mathcal{A} by adding a fresh state q with a self-loop labeled by $(\#, \vec{0})$ as well as transitions labeled by $(\#, \vec{0})$ leading from the accepting states of \mathcal{A} to q . Here, $\#$ is a fresh letter and $\vec{0}$ is the zero vector of the correct dimension. By declaring q to be the only accepting state in \mathcal{A}' , we have that $L_R^s(\mathcal{A}', C)$ is nonempty if and only if $L(\mathcal{A}, C)$ is nonempty.

Note that hardness holds already for deterministic automata, as one can always rename letters to make a nondeterministic PA deterministic without changing the answer to the emptiness problem.

2. Due to Theorem 7.3, it is enough to consider synchronous Büchi acceptance for the upper bound. So, fix some sBPA (\mathcal{A}, C) with $\mathcal{A} = (Q, \Sigma \times D, q_I, \Delta, F)$. Let $C = \bigcup_i L_i$ where the L_i are linear sets. The language $L_B^s(\mathcal{A}, C)$ is nonempty if and only if $L_B^s(\mathcal{A}, L_i)$ is nonempty for some i . Hence, we show how to solve emptiness for automata with linear C , say $C = \left\{ \vec{v}_0 + \sum_{i=1}^k c_i \vec{v}_i \mid c_i \in \mathbb{N} \text{ for } i = 1, \dots, k \right\}$. We define $P = \left\{ \sum_{i=1}^k c_i \vec{v}_i \mid c_i \in \mathbb{N} \text{ for } i = 1, \dots, k \right\}$ and, for a given state $q \in Q$, the NFA

- \mathcal{A}_q obtained from \mathcal{A} by replacing the set of accepting states by $\{q\}$, and
- $\mathcal{A}_{q,q}$ obtained from \mathcal{A} by replacing the initial state by q , by replacing the set of accepting states by $\{q\}$, and by modifying the resulting NFA such that it does not accept the empty word (but leaving its language unchanged otherwise).

We claim that $L_B^s(\mathcal{A}, C)$ is nonempty if and only if there is a $q \in F$ such that both $L(\mathcal{A}_q, C)$ and $L(\mathcal{A}_{q,q}, P)$ are nonempty. As emptiness of Parikh automata is in NP, this yields the desired upper bound.

So, assume there is such a q . Then, there is a finite run ρ_1 of \mathcal{A} that starts in q_I , ends in q , and processes some $w_1 \in \Sigma^*$ with extended Parikh image in C . Also, there is a finite run ρ_2 of \mathcal{A} that starts and ends in q and processes some nonempty $w_2 \in \Sigma^*$ with extended Parikh image in P . For every $n \geq 1$, $\rho_1(\rho_2)^n$ is a finite run of (\mathcal{A}, C) ending in the accepting state q that processes $w_1(w_2)^n$ and whose extended Parikh image is in C . So, $\rho_1(\rho_2)^\omega$ is a synchronous Büchi accepting run of (\mathcal{A}, C) .

For the converse direction, assume that there is some synchronous Büchi accepting run $(q_0, w_0, q_1)(q_1, w_1, q_2)(q_2, w_2, q_3) \cdots$ of (\mathcal{A}, C) . Then, there is also an accepting state $q \in F$ and an infinite set of positions $S \subseteq \mathbb{N}$ such that $q_s = q$ and $\Phi_e(w_0 \cdots w_{s-1}) \in C$ for all $s \in S$. Hence, for every $s \in S$ there is a vector $(c_1^s, \dots, c_k^s) \in \mathbb{N}^k$ such that $\Phi_e(w_0 \cdots w_{s-1}) = \vec{v}_0 + \sum_{i=1}^k c_i^s \vec{v}_i$. By Dickson's Lemma [10], there are $s_1 < s_2$ such that $c_j^{s_1} \leq c_j^{s_2}$ for every $1 \leq j \leq k$. Then, $\Phi_e(w_{s_1} \cdots w_{s_2-1}) = \sum_{i=1}^k (c_i^{s_2} - c_i^{s_1}) \vec{v}_i$, which implies $\Phi_e(w_{s_1} \cdots w_{s_2-1}) \in P$. Thus, the prefix of ρ of length s_1 is an accepting run of the PA (\mathcal{A}_q, C) and the next $(s_2 - s_1)$ transitions of ρ form a nonempty accepting run of the PA $(\mathcal{A}_{q,q}, P)$.

The NP lower bound follows from the proof of Theorem 13.1 by noticing that we also have that $L_B^s(\mathcal{A}', C)$ is nonempty if and only if $L(\mathcal{A}, C)$ is nonempty.

3. and 4. The two undecidability proofs, based on reductions from undecidable problems for two-counter machines, are relegated to the appendix, which introduces all the required technical details on such machines. ◀

We conclude Subsection 6.1 by displaying an interesting consequence of the decomposition used in the proof of Theorem 13.2: we show that the language of every BPA (on infinite words) can be expressed by combining the languages of well-chosen PA (on finite words). This is similar to what happens in other settings: For instance, ω -regular languages are exactly the languages of the form $\bigcup_{j=1}^n L_j \cdot (L'_j)^\omega$, where each L_j, L'_j is regular. Analogously, ω -context-free languages can be characterized by context-free languages [7]. For Büchi Parikh automata, one direction of the above characterization holds:

► **Lemma 14.** *If a language L is accepted by a BPA then $L = \bigcup_{j=1}^n L_j \cdot (L'_j)^\omega$, where each L_j, L'_j is accepted by some PA.*

Proof. Let L be accepted by an sBPA (\mathcal{A}, C) with $C = \bigcup_{j \in J} C_j$ for some finite set J , where each C_j is a linear set. In the proof of Theorem 13.2, we have defined the NFA \mathcal{A}_q and $\mathcal{A}_{q,q}$ for every state q of \mathcal{A} .

Now, consider some $w \in L$ and an accepting run $\rho = (q_0, w_0, q_1)(q_1, w_1, q_2)(q_2, w_2, q_3) \cdots$ of (\mathcal{A}, C) processing w . Then, there is an accepting state q of \mathcal{A} , some C_j , and an infinite set $S \subseteq \mathbb{N}$ of positions such that $q_s = q$ and $\Phi_e(w_0 \cdots w_{s-1}) \in C_j$ for all $s \in S$. Let $C_j = \left\{ \vec{v}_0 + \sum_{i=1}^k c_i \vec{v}_i \mid c_i \in \mathbb{N} \text{ for } i = 1, \dots, k \right\}$ and let $P_j = \left\{ \sum_{i=1}^k c_i \vec{v}_i \mid c_i \in \mathbb{N} \text{ for } i = 1, \dots, k \right\}$. As before, for every $s \in S$, there is a vector $(c_1^s, \dots, c_k^s) \in \mathbb{N}^k$ such that $\Phi_e(w_0 \cdots w_{s-1}) = \vec{v}_0 + \sum_{i=1}^k c_i^s \vec{v}_i$.

Now, we apply an equivalent formulation of Dickson's Lemma [10], which yields an infinite subset $S' \subseteq S$ with $c_j^s \leq c_j^{s'}$ for all $1 \leq j \leq k$ and all $s, s' \in S'$ with $s < s'$, i.e., we have an increasing chain in S . Let $s_0 < s_1 < s_2 < \cdots$ be an enumeration of S' .

As above, $\Phi_e(w_{s_n} \cdots w_{s_{n+1}-1}) \in P_j$ for all n . So, $(q_0, w_0, q_1) \cdots (q_{s_0-1}, w_{s_0-1}, q_{s_0})$ is an accepting run of the PA (\mathcal{A}_q, C) and each $(q_{s_n}, w_{s_n}, q_{s_n+1}) \cdots (q_{s_{n+1}-1}, w_{s_{n+1}-1}, q_{s_{n+1}})$ is an accepting run of the PA $(\mathcal{A}_{q,q}, P)$. So, $w \in \bigcup_{j \in J} \bigcup_{q \in Q} L(\mathcal{A}_q, C_j) \cdot (L(\mathcal{A}_{q,q}, P_j))^\omega$. ◀

Recall that a word is ultimately periodic if it is of the form xy^ω . Every nonempty ω -regular and every nonempty ω -context-free language contains an ultimately periodic word, which is a simple consequence of them being of the form $\bigcup_{j=1}^n L_j \cdot (L'_j)^\omega$.

► **Corollary 15.** *Every nonempty language accepted by a BPA contains an ultimately periodic word.*

Let us briefly comment on the other direction of the implication stated in Lemma 14, i.e., is every language of the form $\bigcup_{j=1}^n L_j \cdot (L'_j)^\omega$, where each L_j, L'_j is accepted by some PA, also accepted by some BPA? The answer is no: Consider $L = \{a^n b^n \mid n > 1\}$, which is accepted by a deterministic PA. However, using the shifting technique (see Remark 3), one can show that L^ω is not accepted by any BPA: Every accepting run of an n -state BPA processing $(a^n b^n)^\omega$ can be turned into an accepting run on a word of the form $(a^n b^n)^* a^{n+k} b^n (a^n b^n)^* a^{n-k} b^n (a^n b^n)^\omega$ for some $k > 0$ by shifting some cycle to the front while preserving Büchi acceptance.

For reachability acceptance, a similar characterization holds, as every RPA can be turned into an equivalent BPA. But for safety and co-Büchi acceptance the characterization question is nontrivial, as for these acceptance conditions all (almost all) run prefixes have to be FC-prefixes. We leave this problem for future work.

6.2 Universality

Now, we consider the universality problem. Here, the positive results follow from the duality of deterministic SPA and RPA (CPA and BPA) and the decidability of nonemptiness for the dual automata classes. Similarly, the undecidability proofs for deterministic sRPA and sBPA follow from duality and undecidability of nonemptiness for the dual automata classes. Finally, the remaining undecidability results follow from reductions from undecidable problems for two-counter machines and Parikh automata over finite words.

► **Theorem 16.** *The following problems are coNP-complete:*

1. *Given a deterministic SPA, is its language universal?*
2. *Given a deterministic CPA, is its language universal?*

The following problems are undecidable:

3. *Given a deterministic sRPA, is its language universal?*
4. *Given an SPA, is its language universal?*
5. *Given a deterministic sBPA, is its language universal?*
6. *Given a CPA, is its language universal?*

Proof. The proofs of the results for deterministic automata follow immediately from the fact that a language is universal if and only if its complement is empty, Theorem 10, and Theorem 13. The proof of undecidability for SPA, based on a reduction from the termination problem for two-counter machines, is relegated to the appendix where all the necessary technical details are presented. To conclude, let us consider universality of CPA.

6.) Universality of Parikh automata over finite words is undecidable [19]. Now, given a PA (\mathcal{A}, C) over Σ , one can construct a CPA (\mathcal{A}', C') for the language $L(\mathcal{A}, C) \cdot \# \cdot (\Sigma \cup \{\#\})^\omega \cup \Sigma^\omega$, where $\# \notin \Sigma$ is a fresh letter. This construction relies on freezing the counters (i.e., moving to a copy of the automaton with the same transition structure, but where the counters are no longer updated) and closure of CPA under union. Now, $L(\mathcal{A}, C)$ is universal if and only if $L_C(\mathcal{A}', C')$ is universal. ◀

6.3 Model Checking

Model checking is arguably the most successful application of automata theory to automated verification. The problem asks whether a given system satisfies a specification, often given by an automaton.

More formally, and for the sake of notational convenience, we say that a transition system \mathcal{T} is a (possibly incomplete) SPA (\mathcal{A}, C) so that every state of \mathcal{A} is accepting and $C = \mathbb{N}^d$, i.e., every run is accepting. Now, the model-checking problem for a class \mathcal{L} of languages of infinite words asks, given a transition system \mathcal{T} and a language $L \in \mathcal{L}$, whether $L_S(\mathcal{T}) \subseteq L$, i.e., whether every word in the transition system satisfies the specification L . Note that our definition here is equivalent to the standard definition of model checking of finite-state transition systems.

Here, we study the model-checking problem for different types of Parikh automata.

► **Theorem 17.** *The following problems are coNP-complete:*

1. *Given a transition system \mathcal{T} and a deterministic SPA (\mathcal{A}, C) , is $L_S(\mathcal{T}) \subseteq L_S(\mathcal{A}, C)$?*
2. *Given a transition system \mathcal{T} and a deterministic CPA (\mathcal{A}, C) , is $L_S(\mathcal{T}) \subseteq L_C(\mathcal{A}, C)$?*

The following problems are undecidable:

3. *Given a transition system \mathcal{T} and a deterministic sRPA (\mathcal{A}, C) , is $L_S(\mathcal{T}) \subseteq L_R^s(\mathcal{A}, C)$?*
4. *Given a transition system \mathcal{T} and an SPA (\mathcal{A}, C) , is $L_S(\mathcal{T}) \subseteq L_S(\mathcal{A}, C)$?*
5. *Given a transition system \mathcal{T} and a deterministic sBPA (\mathcal{A}, C) , is $L_S(\mathcal{T}) \subseteq L_B^s(\mathcal{A}, C)$?*
6. *Given a transition system \mathcal{T} and a CPA (\mathcal{A}, C) , is $L_S(\mathcal{T}) \subseteq L_C(\mathcal{A}, C)$?*

Proof. Let \mathcal{T} be a transition system with $L_S(\mathcal{T}) = \Sigma^\omega$, e.g., a one-state transition system with a self-loop labeled with all letters in Σ . Then, $L \subseteq \Sigma^\omega$ is universal if and only if $L \cap L_S(\mathcal{T}) \subseteq L$. Thus, all six lower bounds (coNP-hardness and undecidability) immediately follow from the analogous lower bounds for universality (see Theorem 16). So, it remains to consider the two coNP upper bounds.

So, fix a deterministic SPA (\mathcal{A}, C) and a transition system \mathcal{T} . We apply the usual approach to automata-theoretic model checking: We have $L_S(\mathcal{T}) \subseteq L_S(\mathcal{A}, C)$ if and only if $L_S(\mathcal{T}) \cap \overline{L_S(\mathcal{A}, C)} = \emptyset$. Due to Theorem 10.3 there is a deterministic sRPA (\mathcal{A}', C') accepting $\overline{L_S(\mathcal{A}, C)}$. Furthermore, using a product construction, one can construct an RPA (\mathcal{A}'', C'') accepting $L_S(\mathcal{T}) \cap \overline{L_S(\mathcal{A}, C)}$, which can then be tested for emptiness, which is in coNP (see Theorem 13). Note that the product construction depends on the fact that every run of \mathcal{T} is accepting, i.e., the acceptance condition of the product automaton only has to check one acceptance condition.

The proof for deterministic co-Büchi Parikh automata is analogous, but using Büchi automata (\mathcal{A}', C') and (\mathcal{A}'', C'') . ◀

6.4 Infinite Games

In this section, we study infinite games with winning conditions specified by Parikh automata. Such games are the technical core of the synthesis problem, the problem of determining whether there is a reactive system satisfying a given specification on its input-output behavior. Our main result is that solving infinite games is undecidable for all acceptance conditions we consider here.

Here, we consider Gale-Stewart games [15], abstract games induced by a language L of infinite words, in which two players alternately pick letters, thereby constructing an infinite word w . One player aims to ensure that w is in L while the other aims to ensure that it is not in L . Formally, given a language $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, the game $G(L)$ is played between Player 1 and Player 2 in rounds $i = 0, 1, 2, \dots$ as follows: At each round i , first Player 1 plays a letter $a_i \in \Sigma_1$ and then Player 2 answers with a letter $b_i \in \Sigma_2$. A play of $G(L)$ is an infinite outcome $w = (a_0, b_0)(a_1, b_1) \dots$ and Player 2 wins it if and only if $w \in L$.

A strategy for Player 2 in $G(L)$ is a mapping from Σ_1^+ to Σ_2 that gives for each prefix played by Player 1 the next letter to play. An outcome $(a_0, b_0)(a_1, b_1) \dots$ agrees with a strategy σ if for each i , we have that $b_i = \sigma(a_0 a_1 \dots a_i)$. Player 2 wins $G(L)$ if she has a strategy that only agrees with outcomes that are winning for Player 2.

The next result follows immediately from the fact that for all classes of deterministic Parikh automata, either nonemptiness or universality is undecidable, and that these two problems can be reduced to solving Gale-Stewart games.

► **Theorem 18.** *The problem “Given an automaton (\mathcal{A}, C) , does Player 2 win $G(L(\mathcal{A}, C))$?” is undecidable for the following classes of automata: (deterministic) sRPA, (deterministic) SPA, (deterministic) sBPA, and (deterministic) CPA.*

Proof. The results follow immediately from the following two facts and the undecidability of emptiness or universality for the corresponding automata types. Fix a language L .

- Player 2 wins $G(\binom{\#}{L})$ with $\binom{\#}{L} = \{(\binom{\#}{w_0})(\binom{\#}{w_1})(\binom{\#}{w_2}) \dots \mid w_0 w_1 w_2 \dots \in L\}$ if and only if L is nonempty.
- Player 2 wins $G(\binom{L}{\#})$ with $\binom{L}{\#} = \{(\binom{w_0}{\#})(\binom{w_1}{\#})(\binom{w_2}{\#}) \dots \mid w_0 w_1 w_2 \dots \in L\}$ if and only if L is universal.

To conclude, note that a Parikh automaton for L can be turned into an equivalent one for $\binom{\#}{L}$ and $\binom{L}{\#}$ while preserving determinism and the acceptance type, by just replacing each transition label a by $\binom{\#}{a}$ and $\binom{a}{\#}$, respectively. ◀

7 Conclusion

In this work, we have extended Parikh automata to infinite words and studied expressiveness, closure properties, and decision problems. Unlike their ω -regular counterparts, Parikh automata on infinite words do not form a nice hierarchy induced by their acceptance conditions. This is ultimately due to the fact that transitions cannot be disabled by the counters running passively along a run. Therefore, a safety condition on the counters cannot be turned into a, say, Büchi condition on the counters, something that is trivial for state conditions. Furthermore, we have shown that emptiness, universality, and model checking are decidable for some of the models we introduced, but undecidable for others. Most importantly, we prove coNP-completeness of model checking with specifications given by deterministic Parikh automata with safety and co-Büchi acceptance. This allows for the automated verification of quantitative safety and persistence properties. Finally, solving infinite games is undecidable for all models.

Note that we have “only” introduced reachability, safety, Büchi, and co-Büchi Parikh automata. There are many more acceptance conditions in the ω -regular setting, e.g., parity, Rabin, Streett, and Muller. We have refrained from generalizing these, as any natural definition of these acceptance conditions will subsume co-Büchi acceptance, and therefore have an undecidable nonemptiness problem.

In future work, we aim to close the open closure property in Table 1. Also, we leave open the complexity of the decision problems in case the semilinear sets are not given by their generators, but by a Presburger formula.

One of the appeals of Parikh automata over finite words is their robustness: they can equivalently be defined via a quantitative variant of WMSO, via weighted automata, and other models (see the introduction for a more complete picture). In future work, we aim to provide similar alternative definitions for Parikh automata on infinite words, in particular, comparing our automata to blind multi-counter automata [11] and reversal-bounded counter machines [18]. However, let us mention that the lack of closure properties severely limits the chances for a natural fragment of MSO being equivalent to Parikh automata on infinite words.

Let us conclude with the following problem for further research: If a Parikh automaton with, say safety acceptance, accepts an ω -regular language, is there then an equivalent ω -regular safety automaton? Stated differently, does Parikhness allow to accept more ω -regular languages? The same question can obviously be asked for other acceptance conditions as well.

References

- 1 Vincent D. Blondel, Olivier Bournez, Pascal Koiran, Christos H. Papadimitriou, and John N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theor. Comput. Sci.*, 255(1-2):687–696, 2001. doi:10.1016/S0304-3975(00)00399-6.
- 2 Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. Weakly-unambiguous Parikh automata and their link to holonomic series. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020*, volume 168 of *LIPICs*, pages 114:1–114:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.114.
- 3 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of Parikh automata and related models. In Rudolf Freund, Markus Holzer, Carlo Mereghetti, Friedrich Otto, and Beatrice Palano, editors, *NCMA 2011*, volume 282 of *books@ocg.at*, pages 103–119. Austrian Computer Society, 2011.

- 4 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.
- 5 Giusi Castiglione and Paolo Massazza. On a class of languages with holonomic generating functions. *Theor. Comput. Sci.*, 658:74–84, 2017. doi:10.1016/j.tcs.2016.07.022.
- 6 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPICs*, pages 117:1–117:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.117.
- 7 Rina S. Cohen and Arie Y. Gold. Theory of omega-languages. I. Characterizations of omega-context-free languages. *J. Comput. Syst. Sci.*, 15(2):169–184, 1977. doi:10.1016/S0022-0000(77)80004-4.
- 8 Luc Dartois, Emmanuel Filiot, and Jean-Marc Talbot. Two-way Parikh automata with a visibly pushdown stack. In Mikolaj Bojanczyk and Alex Simpson, editors, *FOSSACS 2019*, volume 11425 of *LNCS*, pages 189–206. Springer, 2019. doi:10.1007/978-3-030-17127-8_11.
- 9 Jürgen Dassow and Victor Mitrana. Finite automata over free groups. *Int. J. Algebra Comput.*, 10(6):725–738, 2000. doi:10.1142/S0218196700000315.
- 10 Leonard E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. Journal Math.*, 35(4):413–422, 1913.
- 11 Henning Fernau and Ralf Stiebe. Blind counter automata on omega-words. *Fundam. Informaticae*, 83(1-2):51–64, 2008. URL: <http://content.iospress.com/articles/fundamenta-informaticae/fi83-1-2-06>.
- 12 Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *LICS 2015*, pages 329–340. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.39.
- 13 Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-way Parikh automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *FSTTCS 2019*, volume 150 of *LIPICs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.FSTTCS.2019.40.
- 14 Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. *Int. J. Found. Comput. Sci.*, 31(6):711–748, 2020. doi:10.1142/S0129054120410038.
- 15 David Gale and F. M. Stewart. Infinite games with perfect information. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, chapter 13, pages 245–266. Princeton University Press, 1953.
- 16 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. doi:pjm/1102994974.
- 17 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. Parikh automata over infinite words. *arXiv*, 2207.07694, 2022. doi:10.48550/arXiv.2207.07694.
- 18 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978. doi:10.1145/322047.322058.
- 19 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP 2003*, volume 2719 of *LNCS*, pages 681–696. Springer, 2003. doi:10.1007/3-540-45061-0_54.
- 20 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
- 21 Victor Mitrana and Ralf Stiebe. Extended finite automata over groups. *Discret. Appl. Math.*, 108(3):287–300, 2001. doi:10.1016/S0166-218X(00)00200-6.
- 22 J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966. doi:10.1016/S0049-237X(09)70564-6.
- 23 Karianto Wong. Parikh automata with pushdown stack, 2004. Diploma thesis, RWTH Aachen University. URL: <https://old.automata.rwth-aachen.de/download/papers/karianto/ka04.pdf>.

A Appendix: Parikh Automata and Two-counter Machines

Many of our undecidability proofs are reductions from nontermination problems for two-counter machines. To simulate these machines by Parikh automata, we require them to be in a certain normal form. We first introduce two-counter machines, then the normal form, and conclude this section by presenting the simulation via Parikh automata.

A two-counter machine \mathcal{M} is a sequence

$$(0 : I_0)(1 : I_1) \cdots (k-2 : I_{k-2})(k-1 : \text{STOP}),$$

where the first element of a pair $(\ell : I_\ell)$ is the line number and I_ℓ for $0 \leq \ell < k-1$ is an instruction of the form

- $\text{INC}(X_i)$ with $i \in \{0, 1\}$,
- $\text{DEC}(X_i)$ with $i \in \{0, 1\}$, or
- $\text{IF } X_i=0 \text{ GOTO } \ell' \text{ ELSE GOTO } \ell''$ with $i \in \{0, 1\}$ and $\ell', \ell'' \in \{0, \dots, k-1\}$.

A configuration of \mathcal{M} is of the form (ℓ, c_0, c_1) with $\ell \in \{0, \dots, k-1\}$ (the current line number) and $c_0, c_1 \in \mathbb{N}$ (the current contents of the counters). The initial configuration is $(0, 0, 0)$ and the unique successor configuration of a configuration (ℓ, c_0, c_1) is defined as follows:

- If $I_\ell = \text{INC}(X_i)$, then the successor configuration is $(\ell + 1, c'_0, c'_1)$ with $c'_i = c_i + 1$ and $c'_{1-i} = c_{1-i}$.
- If $I_\ell = \text{DEC}(X_i)$, then the successor configuration is $(\ell + 1, c'_0, c'_1)$ with $c'_i = \max\{c_i - 1, 0\}$ and $c'_{1-i} = c_{1-i}$.
- If $I_\ell = \text{IF } X_i=0 \text{ GOTO } \ell' \text{ ELSE GOTO } \ell''$ and $c_i = 0$, then the successor configuration is (ℓ', c_0, c_1) .
- If $I_\ell = \text{IF } X_i=0 \text{ GOTO } \ell' \text{ ELSE GOTO } \ell''$ and $c_i > 0$, then the successor configuration is (ℓ'', c_0, c_1) .
- If $I_\ell = \text{STOP}$, then (ℓ, c_0, c_1) has no successor configuration.

The unique run of \mathcal{M} (starting in the initial configuration) is defined as expected. It is either finite (line $k-1$ is reached) or infinite (line $k-1$ is never reached). In the former case, we say that \mathcal{M} terminates.

► **Proposition 19** ([20]). *The following problem is undecidable: Given a two-counter machine \mathcal{M} , does \mathcal{M} terminate?*

In the following, we assume without loss of generality that each two-counter machine satisfies the *guarded-decrement property*: Every decrement instruction $(\ell : \text{DEC}(X_i))$ is preceded by $(\ell-1 : \text{IF } X_i=0 \text{ GOTO } \ell+1 \text{ ELSE GOTO } \ell)$ and decrements are never the target of a goto instruction. As the decrement of a zero counter has no effect, one can modify each two-counter machine \mathcal{M} into an \mathcal{M}' satisfying the guarded-decrement property such that \mathcal{M} terminates if and only if \mathcal{M}' terminates: One just adds the the required guard before every decrement instruction and changes each target of a goto instruction that is a decrement instruction to the preceding guard.

The guarded-decrement property implies that decrements are only executed if the corresponding counter is nonzero. Thus, the value of counter i after a finite sequence of executed instructions (starting with value zero in the counters) is equal to the number of executed increments of counter i minus the number of executed decrements of counter i . Note that the number of executed increments and decrements can be tracked by a Parikh automaton.

Consider a finite or infinite word $w = w_0w_1w_2 \cdots$ over the set $\{0, 1, \dots, k-1\}$ of line numbers. We now describe how to characterize whether w is (a prefix of) the projection to the line numbers of the unique run of \mathcal{M} starting in the initial configuration. This

characterization is designed to be checkable by a Parikh automaton. Note that w only contains line numbers, but does not encode values of the counters. These will be kept track of by the Parikh automaton by counting the number of increment and decrement instructions in the input, as explained above (this explains the need for the guarded-decrement property). Formally, we say that w contains an *error* at position $n < |w| - 1$ if either $w_n = k - 1$ (the instruction in line w_n is STOP), or if one of the following two conditions is satisfied:

1. The instruction I_{w_n} in line w_n of \mathcal{M} is an increment or a decrement and $w_{n+1} \neq w_n + 1$, i.e., the letter w_{n+1} after w_n is not equal to the line number $w_n + 1$, which it should be after an increment or decrement.
2. I_{w_n} has the form IF $X_i=0$ GOTO ℓ ELSE GOTO ℓ' , and one of the following cases holds: Either, we have

$$\sum_{j: I_j=\text{INC}(X_i)} |w_0 \cdots w_n|_j = \sum_{j: I_j=\text{DEC}(X_i)} |w_0 \cdots w_n|_j$$

and $w_{n+1} \neq \ell$, i.e., the number of increments of counter i is equal to the number of decrements of counter i in $w_0 \cdots w_n$ (i.e., the counter is zero) but the next line number in w is not the target of the if-branch. Or, we have

$$\sum_{j: I_j=\text{INC}(X_i)} |w_0 \cdots w_n|_j \neq \sum_{j: I_j=\text{DEC}(X_i)} |w_0 \cdots w_n|_j,$$

and $w_{n+1} \neq \ell'$, i.e., the number of increments of counter i is not equal to the number of decrements of counter i in $w_0 \cdots w_n$ (i.e., the counter is nonzero) but the next line number in w is not the target of the else-branch.

Note that the definition of error (at position n) refers to the number of increments and decrements in the prefix $w_0 \cdots w_n$, which does not need to be error-free itself. However, if a sequence of line numbers does not have an error, then the guarded-decrement property yields the following result.

► **Lemma 20.** *Let $w \in \{0, 1, \dots, k-1\}^+$ with $w_0 = 0$. Then, w has no errors at positions $\{0, 1, \dots, |w| - 2\}$ if and only if w is a prefix of the projection to the line numbers of the run of \mathcal{M} .*

Proof. If w has no errors at positions $\{0, 1, \dots, |w| - 2\}$, then an induction shows that (w_n, c_0^n, c_1^n) with

$$c_i^n = \sum_{j: I_j=\text{INC}(X_i)} |w_0 \cdots w_{n-1}|_j - \sum_{j: I_j=\text{DEC}(X_i)} |w_0 \cdots w_{n-1}|_j$$

is the n -th configuration of the run of \mathcal{M} .

On the other hand, projecting a prefix of the run of \mathcal{M} to the line numbers yields a word w without errors at positions $\{0, 1, \dots, |w| - 2\}$. ◀

The existence of an error can be captured by a Parikh automaton, leading to the undecidability of the safe word problem for Parikh automata, which we now prove. Let (\mathcal{A}, C) be a PA accepting finite words over Σ . A *safe* word of (\mathcal{A}, C) is an infinite word in Σ^ω such that each of its prefixes is in $L(\mathcal{A}, C)$.

► **Lemma 21.** *The following problem is undecidable: Given a deterministic PA, does it have a safe word?*

Proof. Our proof proceeds by a reduction from the nontermination problem for decrement-guarded two-counter machines. Given such a machine $\mathcal{M} = (0 : I_0) \cdots (k-2 : I_{k-2})(k-1 : \text{STOP})$ let $\Sigma = \{0, \dots, k-1\}$ be the set of its line numbers. We construct a deterministic PA $(\mathcal{A}_{\mathcal{M}}, C_{\mathcal{M}})$ that accepts a word $w \in \Sigma^*$ if and only if $w = \varepsilon$, $w = 0$, or if $|w| \geq 2$ and w

does not contain an error at position $|w| - 2$ (but might contain errors at earlier positions). Intuitively, the automaton checks whether the second-to-last instruction is executed properly. The following is then a direct consequence of Lemma 20: $(\mathcal{A}_{\mathcal{M}}, C_{\mathcal{M}})$ has a safe word if and only if \mathcal{M} does not terminate.

The deterministic PA $(\mathcal{A}_{\mathcal{M}}, C_{\mathcal{M}})$ keeps track of the occurrence of line numbers with increment and decrement instructions of each counter (using four dimensions) and two auxiliary dimensions to ensure that the two cases in Condition 2 of the error definition on Page 17 are only checked when the second-to-last letter corresponds to a goto instruction. More formally, we construct $\mathcal{A}_{\mathcal{M}}$ such the unique run processing some input $w = w_0 \dots w_{n-1}$ has the extended Parikh image $(v_{\text{inc}}^0, v_{\text{dec}}^0, v_{\text{goto}}^0, v_{\text{inc}}^1, v_{\text{dec}}^1, v_{\text{goto}}^1)$ where

- v_{inc}^i is equal to $\sum_{j: \text{I}_{w_{n-2}} = \text{INC}(x_i)} |w_0 \dots w_{n-2}|_j$, i.e., the number of increment instructions read so far (ignoring the last letter),
- v_{dec}^i is equal to $\sum_{j: \text{I}_{w_{n-2}} = \text{DEC}(x_i)} |w_0 \dots w_{n-2}|_j$, i.e., the number of decrement instructions read so far (ignoring the last letter), and
- $v_{\text{goto}}^i \bmod 4 = 0$, if the second-to-last instruction $\text{I}_{w_{n-2}}$ is not a goto testing counter i ,
- $v_{\text{goto}}^i \bmod 4 = 1$, if the second-to-last instruction $\text{I}_{w_{n-2}}$ is a goto testing counter i and the last letter w_{n-1} is equal to the target of the if-branch of this instruction, and
- $v_{\text{goto}}^i \bmod 4 = 2$, if the second-to-last instruction $\text{I}_{w_{n-2}}$ is a goto testing counter i and the last letter w_{n-1} is equal to the target of the else-branch of this instruction.
- $v_{\text{goto}}^i \bmod 4 = 3$, if the second-to-last instruction $\text{I}_{w_{n-2}}$ is a goto testing counter i and the last letter w_{n-1} is neither equal to the target of the if-branch nor equal to the target of the else-branch of this instruction. Note that this constitutes an error at position $n - 2$.

Note that $v_{\text{goto}}^i \bmod 4 \neq 0$ can be true for at most one of the i at any time (as a goto instruction $\text{I}_{w_{n-2}}$ only refers to one counter) and that the v_{inc}^i and v_{dec}^i are updated with a delay of one transition (as the last letter of w is ignored). This requires to store the previously processed letter in the state space of $\mathcal{A}_{\mathcal{M}}$.

Further, $C_{\mathcal{M}}$ is defined such that $(v_{\text{inc}}^0, v_{\text{dec}}^0, v_{\text{goto}}^0, v_{\text{inc}}^1, v_{\text{dec}}^1, v_{\text{goto}}^1)$ is in $C_{\mathcal{M}}$ if and only if

- $v_{\text{goto}}^i \bmod 4 = 0$ for both i , or if
- $v_{\text{goto}}^i \bmod 4 = 1$ for some i (recall that i is unique then) and $v_{\text{inc}}^i = v_{\text{dec}}^i$, or if
- $v_{\text{goto}}^i \bmod 4 = 2$ for some i (again, i is unique) and $v_{\text{inc}}^i \neq v_{\text{dec}}^i$.

All other requirements, e.g., Condition 1 of the error definition on Page 17, the second-to-last letter not being $k - 1$, and the input being in $\{\varepsilon, 0\}$, can be checked using the state space of $\mathcal{A}_{\mathcal{M}}$. ◀

A.1 Proofs omitted in Section 6

First, we prove that nonemptiness for deterministic SPA is undecidable.

Proof of Theorem 13.3. The result follows immediately from Lemma 21: A PA (\mathcal{A}, C) has a safe word if and only if $L_S(\mathcal{A}, C) \neq \emptyset$. ◀

Next, we prove undecidability of nonemptiness for deterministic CPA.

Proof of Theorem 13.4. We present a reduction from the universal termination problem [1]¹ for decrement-guarded two-counter machines, which is undecidable. The problem asks whether a given two-counter machine \mathcal{M} terminates from every configuration. If this is the case, we say that \mathcal{M} is universally terminating.

¹ The authors use a slightly different definition of two-counter machine than we do here. Nevertheless, the universal termination problem for their machines can be reduced to the universal termination problem for decrement-guarded two-counter machines as defined here.

Now, consider a decrement-guarded two-counter machine \mathcal{M} that contains, without loss of generality, an increment instruction for each counter, say in lines ℓ_0^+ and ℓ_1^+ . In the proof of Lemma 21, we construct a deterministic PA $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ that accepts a finite word w over the line numbers of \mathcal{M} if and only if $w = \varepsilon$, $w = 0$, or if $|w| \geq 2$ and w does not contain an error at position $|w| - 2$. We claim that $L_C(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ is nonempty if and only if \mathcal{M} is not universally terminating.

So, first assume there is some $w \in L_C(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$. Hence, there is a run of $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ processing w satisfying the co-Büchi acceptance condition: From some point n_0 onward, the run only visits states in F and the extended Parikh image is in $C_\mathcal{M}$. This means that there is no error in w after position n_0 . So, \mathcal{M} does not terminate from the configuration (w_{n_0}, c_0, c_1) with

$$c_i = \sum_{j: I_j = \text{INC}(x_i)} |w_0 \cdots w_{n_0-1}|_j - \sum_{j: I_j = \text{DEC}(x_i)} |w_0 \cdots w_{n_0-1}|_j,$$

i.e., \mathcal{M} is not universally terminating.

Now, assume \mathcal{M} does not universally terminate, say it does not terminate from configuration (ℓ, c_0, c_1) . Recall that the instruction in line ℓ_i^+ is an increment of counter i . We define $w = (\ell_0^+)^{c_0} (\ell_1^+)^{c_1} w'$ where w' is the projection to the line numbers of the (nonterminating) run of \mathcal{M} starting in (ℓ, c_0, c_1) . This word does not have an error after position ℓ (but may have some before that position). Hence there is a co-Büchi accepting run of $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ processing w , i.e., $L_C(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ is nonempty. \blacktriangleleft

Finally, we prove undecidability of universality for SPA.

Proof of Theorem 16.4. We present a reduction from the termination problem for (decrement-guarded) two-counter machines. So, fix such a machine \mathcal{M} with line numbers $0, 1, \dots, k-1$ where $k-1$ is the line number of the stopping instruction, fix $\Sigma = \{0, 1, \dots, k-1\}$, and consider $L_\mathcal{M} = L_\mathcal{M}^0 \cup L_\mathcal{M}^1$ with

$$\begin{aligned} L_\mathcal{M}^0 &= \{w \in \Sigma^\omega \mid w_0 \neq 0\} \text{ and} \\ L_\mathcal{M}^1 &= \{w \in \Sigma^\omega \mid \text{if } |w|_{k-1} > 0 \text{ then } w \text{ contains an error} \\ &\quad \text{strictly before the first occurrence of } k-1\}. \end{aligned}$$

We first prove that $L_\mathcal{M}$ is not universal if and only if \mathcal{M} terminates, then that $L_\mathcal{M}$ is accepted by some SPA.

So, assume that \mathcal{M} terminates and let $w \in \Sigma^*$ be the projection of the unique finite run of \mathcal{M} to the line numbers. Then, w starts with 0, contains a $k-1$, and no error before the $k-1$. Thus, $w0^\omega$ is not in $L_\mathcal{M}$, i.e., $L_\mathcal{M}$ is not universal.

Conversely, assume that $L_\mathcal{M}$ is not universal. Then, there is an infinite word w that is neither in $L_\mathcal{M}^0$ nor in $L_\mathcal{M}^1$. So, w must start with 0, contain a $k-1$, but no error before the first $k-1$. Thus, Lemma 20 implies that the prefix of w up to and including the first $k-1$ is the projection to the line numbers of the run of \mathcal{M} . This run is terminating, as the prefix ends in $k-1$. Thus, \mathcal{M} terminates.

It remains to argue that $L_\mathcal{M} = L_\mathcal{M}^0 \cup L_\mathcal{M}^1$ is accepted by an SPA. Due to closure of SPA under union and the fact that every ω -regular safety property is also accepted by an SPA, we only need to consider $L_\mathcal{M}^1$. Recall that we have constructed a deterministic PA (\mathcal{A}, C) (on finite words) accepting a word if it is empty, 0, or contains an error at the second-to-last position. We modify this PA into an SPA (\mathcal{A}', C') that accepts $L_\mathcal{M}^1$.

To this end, we add a fresh accepting state q_a and a fresh rejecting state q_r to \mathcal{A} while making all states of \mathcal{A} accepting in \mathcal{A}' . Both fresh states are sinks equipped with self-loops that are labeled with $(\ell, \vec{0})$ for every line number ℓ . Here, $\vec{0}$ is the appropriate zero vector, i.e., the counters are frozen when reaching the fresh states.

40:20 Parikh Automata over Infinite Words

Intuitively, moving to q_a signifies that an error at the current position is guessed. Consequently, if a $k - 1$ is processed from a state of \mathcal{A} , the rejecting sink q_s is reached. We reflect in a fresh component of the extended Parikh image whether q_a has been reached. Due to this, we can define C' so that the extended Parikh image of every run prefix ending in a state of \mathcal{A} is in C' and that the extended Parikh image of a run prefix ending in q_a is in C' if removing the last entry (the reflecting one) yields a vector in C , i.e., an error has indeed occurred. Then, we have $L_S(\mathcal{A}', C') = L_{\mathcal{M}}^1$ as required. ◀

New Analytic Techniques for Proving the Inherent Ambiguity of Context-Free Languages

Florent Koechlin ✉

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

This article extends the work of Flajolet [10] on the relation between generating series and inherent ambiguity. We first propose an analytic criterion to prove the infinite inherent ambiguity of some context-free languages, and apply it to give a purely combinatorial proof of the infinite ambiguity of Shamir’s language. Then we show how Ginsburg and Ullian’s criterion on unambiguous bounded languages translates into a useful criterion on generating series, which generalises and simplifies the proof of the recent criterion of Makarov [21]. We then propose a new criterion based on generating series to prove the inherent ambiguity of languages with interlacing patterns, like $\{a^n b^m a^p b^q \mid n \neq p \text{ or } m \neq q, \text{ with } n, m, p, q \in \mathbb{N}^*\}$. We illustrate the applicability of these two criteria on many examples.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages

Keywords and phrases Inherent ambiguity, Infinite ambiguity, Ambiguity, Generating series, Context-free languages, Bounded languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.41

1 Introduction

A context-free grammar G is said to be *unambiguous* if for any word w recognized by G , there exists exactly one derivation tree for w . A context-free language is called *inherently ambiguous* if it can not be recognized by any unambiguous grammar. Proving that a language is inherently ambiguous is a difficult question, as it is an impossibility notion, and it is undecidable in general [14, 15]. In practice, three different methods have emerged to prove the inherent ambiguity of some context-free languages: an approach based on iterations on derivation trees [25, 26], an other based on iterations on semilinear sets [14, 16, 29], and finally an approach based on generating series [10, 17, 21, 28]. The first two approaches are best suited for (and for the second, limited to) bounded languages.

In this article, we provide new sufficient criteria to prove inherent (infinite) ambiguity, answering two questions of Flajolet [10]. Our main result is an interpretation, in the world of generating series of Ginsburg and Ullian’s criteria [14] on semilinear sets. It rediscovers and generalises the criterion recently developed by Makarov [21], while opening the way to new techniques to prove the inherent ambiguity of unbounded languages or bounded languages with an interlacing pattern. In a different direction, we also provide a criterion for inherent infinite ambiguity.

1.1 Motivation and background

Deciding if a grammar is ambiguous is undecidable [6], as well as deciding if a context-free language is inherently ambiguous [14, 15]. However, detecting ambiguity in context-free grammars has strong implications for compilers and parsers. Therefore, identifying inherently ambiguous languages is an important step towards our understanding of the limits of the model of context-free languages to describe natural or programming languages. Let us start with some context on the methods developed so far to establish the inherent ambiguity of a language.



© Florent Koechlin;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 41; pp. 41:1–41:22



Leibniz International Proceedings in Informatics

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

Bounded languages. The first techniques developed to prove inherent ambiguity dealt with bounded languages. A language L is called *bounded* if there exist words w_1, \dots, w_d with $d \geq 1$ such that $L \subseteq w_1^* \dots w_d^*$. Despite its apparent simplicity, the class of bounded languages is rich enough to provide a large variety of inherently ambiguous languages; furthermore it is often possible to deduce the inherent ambiguity of a context-free language from the inherent ambiguity of a bounded language, using the stability of unambiguous context-free languages under intersection with a regular language [14].

Iteration on derivation trees. In 1961, Parikh was the first to exhibit an inherently ambiguous context-free language, the bounded language $L = \{a^n b^m a^p b^q \mid n = p \text{ or } m = q, \text{ with } n, m, p, q \in \mathbb{N}_{>0}\}$ (see [26] and [27, Theorem 3]). Parikh's proof relies on an iteration argument over the derivation trees of any unambiguous grammar recognising L . A few years later, Ogden generalised this method and published his famous lemma [25], which drastically simplified the identification of iterating pairs in derivation trees. Since then, these iterations techniques have been very popular to study several inherently ambiguous languages (see for instance [7, 24, 30, 32]). However, they remain subtle and difficult to set up in general; hence they are sometimes unsuitable to study complex context-free languages.

Iteration on semilinear sets. In 1966, after Parikh's article but before Ogden's lemma, Ginsburg and Ullian succeeded in using strong iterations arguments on derivation trees to characterise exactly the inherent ambiguity of *bounded* context-free languages in terms of their associated semilinear sets [14]. Their result made it possible to prove the inherent ambiguity of bounded languages using iterations on semilinear sets instead of derivation trees [14, 16, 29]. Unfortunately, iterations on semilinear sets turned out to be almost as laborious as on derivation trees. The simplicity of the proof and the strong applications of Ogden's lemma severely contrasted with Ginsburg and Ullian's criterion¹ that was complex to use and required a lot of case analysis. It may explain why iterations on semilinear sets were supplanted by iterations on derivation trees.

Generating series method. In 1987, Flajolet [10] proposed a conceptually new approach, based on generating series and the contraposition of the Chomsky-Schützenberger theorem [6]. The generating series of a language L is the formal series $\sum_n \ell_n x^n$ where ℓ_n denotes the number of words of length n in L . Flajolet's idea consists in showing that a language is inherently ambiguous by computing its generating series – which is a purely combinatorial question, for which there are many techniques [11] – and showing that this series is not algebraic – for which there are also several mathematical characterisations [10]. This method turned out to be very successful, as Flajolet was able to easily prove the inherent ambiguity of a dozen languages in his article. It complemented very well the previous techniques used for proving ambiguity: whereas iterations arguments are rather efficient and fast for proving the inherent ambiguity of languages with a simple structure, which tend to have an algebraic generating series², on the opposite side, the generating series approach allows to deal with complex languages that have a transcendental (*i.e.* non-algebraic) generating series and seem out of reach of iterations techniques.

¹ In his book [12, p.211], Ginsburg wondered whether there was a simpler technique to prove the inherent ambiguity of $L := \{a^n b^m c^p : n = m \text{ or } m = p\}$, and in a sense Ogden answered in the positive.

² For example, all bounded context-free languages have a rational generating series

Limits. Nevertheless, the three presented approaches sometimes fail on very simple context-free languages expected to be inherently ambiguous, like the language $L' := \{a^n b^m c^p : n \neq m \text{ or } m \neq p\}$. It has a rational – hence algebraic – generating series, and iterations arguments (whether on trees or semilinear sets) struggle to handle the inequality condition that does not constrain anymore the form of iterating pairs. It is not very surprising that those methods do not cover every language, as hinted by the fact that deciding inherent ambiguity is undecidable.

1.2 Problem statement and contributions

At the end of his article [10], Flajolet raised several open questions about the relation between inherent ambiguity and generating series: is it possible to capture the inherent infinite ambiguity of some context-free languages using analytic tools on generating series? Can rational generating series still be useful to prove the inherent ambiguity of languages like $L' = \{a^n b^m c^p : n \neq m \text{ or } m \neq p\}$? Recently, Makarov [21] answered the second question by using new ideas coming from the generating series of $GF(2)$ grammars. He provided a simple criterion on rational series to prove the inherent ambiguity of some bounded languages on $a_1^* \dots a_d^*$, where the a_i 's are distinct letters, and proved the inherent ambiguity of L' .

In this article, we give new answers to the two open questions of Flajolet about inherent ambiguity and infinite inherent ambiguity. We first propose an analytic technique to prove the infinite inherent ambiguity of context-free languages (Theorem 4), and apply it to give a *purely combinatorial* proof of the infinite ambiguity of Shamir's language (Corollary 7). Then we use Ginsburg and Ullian's characterisation to derive a simple criterion (Theorem 12) on generating series to prove the inherent ambiguity of some bounded languages, which both generalises and simplifies the proof of an analogous criterion recently found by [21]. We then propose a new criterion based on generating series to prove the inherent ambiguity of languages with an interlacing pattern, that are not covered by [21], like $L'' = \{a^n b^m a^p b^q \mid n \neq p \text{ or } m \neq q, \text{ with } n, m, p, q \in \mathbb{N}^*\}$ (Theorem 21). To make them amenable to the wider audience possible, these criteria only require a basic knowledge in combinatorics and in polynomials in several variables.

1.3 Related work

To the author's knowledge, since Flajolet's article, and until Makarov's new criterion [21], no real new successful approach based on generating series has been proposed to prove the inherent ambiguity of languages. Several years after Flajolet's article, a subclass of unambiguous context-free language (called slender languages) has been shown to be associated to rational series [17], but their criterion can be in fact interpreted as a shortcut of Flajolet's technique³. More recently [1], the class of generating series associated to unambiguous context-free grammars, called \mathbb{N} -algebraic series, has been precisely described as well as their asymptotic behaviour. This class of generating series does, however, enjoy less closure properties than algebraic series, which makes them less applicable for proving inherent ambiguity.

If the techniques developed in this article are based on generating series and hence lie in the continuity of Flajolet's method [10], they can also be seen as a nice alliance of the three historical techniques presented in this introduction: we use Flajolet's idea to study ambiguity through generating series [10], in order to revisit from this point of view Ginsburg and Ullian's criteria [14], whose proof relies on iterations in derivation trees.

³ By [2], if the generating series of a slender language is not rational then it is also not algebraic

2 Preliminaries

Context-free languages. A context-free grammar (CFG for short) is a tuple $G = (N, \Sigma, S, D)$, where Σ is a finite set of terminal symbols, N is a finite set of non-terminal symbols, $S \in N$ is the axiom, and $D \subseteq N \times (N \cup \Sigma)^*$ is the finite set of derivation rules. A rule $(A, w) \in D$ is usually written $A \rightarrow w$, with $A \in N$ and $w \in (N \cup \Sigma)^*$. The derivation rules of D can be seen as rewriting rules affecting only non terminal symbols. Let w, w' be two words in $(N \cup \Sigma)^*$. The application of a rewriting rule of D to a non-terminal symbol of w is called a derivation step of G from w . If w' is derived from w after one derivation step, we write $w \rightarrow_G w'$. A derivation from w to w' is a (possibly empty if $w = w'$) sequence of consecutive derivation steps $w \rightarrow_G w_1 \rightarrow_G \dots \rightarrow_G w'$, denoted by $w \rightarrow_G^* w'$. As the order of the application of the derivation rules is not canonical, a derivation is rather described as a tree, called a *derivation tree*. For instance, if $D = \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$, the derivations $S \rightarrow AB \rightarrow aB \rightarrow ab$ and $S \rightarrow AB \rightarrow Ab \rightarrow ab$ have the same derivation tree $\begin{array}{c} S \\ \swarrow \quad \searrow \\ A \quad B \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ a \quad b \quad a \quad b \end{array}$ and can be identified. A word is called terminal if it contains only terminal symbols. The language of G , denoted by $\mathcal{L}(G) \subseteq \Sigma^*$, is the set of terminal words that can be derived from the axiom S . The grammar G is said to be *unambiguous* if for any word $w \in \mathcal{L}(G)$, there exists exactly one derivation tree for w . A *context-free language* (CFL) is a language recognized by a CFG. A CFL is called *inherently ambiguous* if it is not recognisable by any unambiguous CFG.

Univariate series. Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of non-negative integer, \mathbb{Q} the set of rational numbers, \mathbb{F}_2 the field with two elements, and K an arbitrary field (in practice, $K = \mathbb{Q}$ or $K = \mathbb{F}_2$ in this article). The set of polynomials with coefficients in K and indeterminate x is denoted by $K[x]$. We denote by $K[[x]]$ the set of formal series with coefficients in K , which is the set of infinite polynomials of the form $\sum_{n \in \mathbb{N}} a_n x^n$, with $a_n \in K$. We recall that $(K[[x]], +, \cdot)$ has a ring structure, with respect to the addition and the Cauchy product. The series $S(x) = \sum_{n \in \mathbb{N}} x^n$ satisfies the equation $(1 - x)S(x) = 1$, and is hence written $\frac{1}{1-x}$. The set $K(x)$ denotes the set of rational fractions, which is formally the set of fractions of the form $p(x)/q(x)$ where p, q are both polynomials in $K[x]$, with $q(x) \neq 0$. A univariate series $f(x) \in K[[x]]$ is *rational* if it satisfies an equation of the form $q(x)f(x) = p(x)$, where $p, q \in K[x]$, $q \neq 0$. In this case $f(x)$ is written $p(x)/q(x)$. It is called *algebraic* over K if there exists a non null polynomial $P(x, Y)$, with coefficients in K , such that $P(x, f(x)) = 0$.

Let $n \in \mathbb{N}$. If L is a language, we define ℓ_n the number of words in L of length n . The generating series $L(x)$ of L is the formal series $L(x) := \sum_{n \in \mathbb{N}} \ell_n x^n \in \mathbb{Q}[[x]]$. If G is a CFG recognising L , we denote by g_n the number of derivation trees of terminal words of length n . If g_n is finite for all $n \in \mathbb{N}$, the generating series of the derivation trees of G , defined by the formal series $G(x) := \sum_{n \in \mathbb{N}} g_n x^n$, is well-defined. In this case, the description of the grammar G translates directly into a polynomial system satisfied by $G(x)$, which implies that $G(x)$ is algebraic over \mathbb{Q} . If G is unambiguous, then $G(x)$ is well-defined and coincides with $L(x)$, so the generating series of an unambiguous CFL is algebraic over \mathbb{Q} : this is the Chomsky-Schützenberger theorem [6]. The subset of series that are the generating series of the derivation trees of a CFG is called the set of \mathbb{N} -algebraic series, and it is strictly included in the set of algebraic series over \mathbb{Q} [1].

Multivariate polynomials. For every $d \in \mathbb{N}_{>0}$, \mathbb{N}^d denotes the set of vectors with d coordinates in \mathbb{N} . A vector $(v_1, \dots, v_d) \in \mathbb{N}^d$ will be freely written in a condensed notation \mathbf{v} . Similarly, the tuple of d variables (x_1, \dots, x_d) is written \mathbf{x} . The notation $K[\mathbf{x}]$ denotes

the ring of multivariate polynomials with indeterminates $\mathbf{x} = (x_1, \dots, x_d)$ and coefficients in K . For $\mathbf{v} = (v_1, \dots, v_d) \in \mathbb{N}^d$, the monomial $x_1^{v_1} \dots x_d^{v_d}$ is written $\mathbf{x}^{\mathbf{v}}$. The total degree of $x_1^{v_1} \dots x_d^{v_d}$ is the number $v_1 + \dots + v_d$. A polynomial is called *homogenous* if its monomials have the same total degree (for instance $x^2 + xy$ is homogenous but $1 + xy$ is not). A polynomial is called *irreducible* if it is non constant and cannot be decomposed as the product of two non constant polynomials. The set $K(\mathbf{x})$ denotes the field of rational fractions of $K[\mathbf{x}]$, that is the set of quotients $p(\mathbf{x})/q(\mathbf{x})$ where $p, q \in K[\mathbf{x}]$ and $q \neq 0$.

► **Remark 1 (Arithmetic of $K[\mathbf{x}]$).** We chose to use as little mathematical notion of $K[\mathbf{x}]$ as possible, to keep our criteria useful for people that are not familiar with multivariate polynomials. To understand the proofs, it is useful to remember that $K[\mathbf{x}]$ is factorial (see for instance [20, Corrolary 2.4 p 183]): any polynomial in $K[\mathbf{x}]$ admits a unique factorization as a product of irreducible polynomials. However, $K[\mathbf{x}]$ is not principal in general (even when $K = \mathbb{Q}$), nor euclidian; in particular, the Bezout identity does not hold anymore. Hence there is no canonical multivariate equivalent to the euclidian division, and similarly there is no canonical partial fraction decomposition.

Multivariate series. We write $K[[\mathbf{x}]]$ for the ring of formal multivariate series with (commutative) indeterminates \mathbf{x} and coefficients in K , which is the set of infinite polynomials of the form

$$\sum_{\mathbf{v} \in \mathbb{N}^d} a_{\mathbf{v}} \mathbf{x}^{\mathbf{v}} := \sum_{v_1, \dots, v_d \in \mathbb{N}^d} a_{v_1, \dots, v_d} x_1^{v_1} \dots x_d^{v_d}, \quad \text{with } a_{\mathbf{v}} \in K \text{ for all } \mathbf{v} \in \mathbb{N}^d.$$

The series $\sum_{n,m} x^n y^m$ satisfies the equation $(1-x)(1-y)S(x,y) = 1$ and hence is written $\frac{1}{(1-x)(1-y)}$. Similarly, for every monomial $\mathbf{x}^{\mathbf{v}}$, the series $\sum_{n \in \mathbb{N}} \mathbf{x}^{n\mathbf{v}}$ is written $\frac{1}{1-\mathbf{x}^{\mathbf{v}}}$; for instance, the series $\sum_{n,m} x^n y^m$ is written $\frac{1}{1-xy}$. A series $f(\mathbf{x})$ is called algebraic over K if there exists a non null multivariate polynomial $P(\mathbf{x}, Y) \in K[\mathbf{x}, Y]$ such that $P(\mathbf{x}, f(\mathbf{x})) = 0$.

Semilinear sets. Let $d \in \mathbb{N}$. A set $L \subseteq \mathbb{N}^d$ is called *linear* if there exists a vector $\mathbf{c} \in \mathbb{N}^d$, and a finite set of vectors $P = \{\mathbf{p}_1, \dots, \mathbf{p}_s\}$, called periods, such that

$$L = \{\mathbf{c} + \lambda_1 \mathbf{p}_1 + \dots + \lambda_s \mathbf{p}_s : \lambda_1, \dots, \lambda_s \in \mathbb{N}\}.$$

We will denote such a set under the condensed form $\mathbf{c} + P^*$. A *semilinear set* $S \subseteq \mathbb{N}^d$ is a finite union of linear sets in \mathbb{N}^d . The generating series of a semilinear set is defined by the multivariate series $S(\mathbf{x}) = \sum_{\mathbf{v} \in S} \mathbf{x}^{\mathbf{v}}$. In the particular case where $S = \mathbf{c} + P^*$ is a linear set, with $P = \{\mathbf{p}_1, \dots, \mathbf{p}_s\}$ a set of linearly independent periods over \mathbb{Q} , then the decomposition of a vector $\mathbf{v} \in S$ under the form $\mathbf{v} = \mathbf{c} + \lambda_1 \mathbf{p}_1 + \dots + \lambda_s \mathbf{p}_s$ is unique, hence:

$$S(\mathbf{x}) = \sum_{\lambda_1, \dots, \lambda_s \in \mathbb{N}^s} \mathbf{x}^{\mathbf{c} + \lambda_1 \mathbf{p}_1 + \dots + \lambda_s \mathbf{p}_s} = \mathbf{x}^{\mathbf{c}} \sum_{\lambda_1 \in \mathbb{N}} (\mathbf{x}^{\mathbf{p}_1})^{\lambda_1} \dots \sum_{\lambda_s \in \mathbb{N}} (\mathbf{x}^{\mathbf{p}_s})^{\lambda_s} = \frac{\mathbf{x}^{\mathbf{c}}}{\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})}.$$

Note that by [8, 18], it is always possible to find a representation of a semilinear S under the form $S = \bigsqcup_{i=1}^r (\mathbf{c}_i + P_i^*)$, where the union is disjoint, and the vectors are linearly independent over \mathbb{Q} in each P_i . Hence the generating series of a semilinear set is rational and can be deduced from such a presentation by $S(\mathbf{x}) = \sum_{i=1}^r \frac{\mathbf{x}^{\mathbf{c}_i}}{\prod_{\mathbf{p} \in P_i} (1 - \mathbf{x}^{\mathbf{p}})}$.

Computing generating series of semilinear sets. In practice, for the examples of this article, we will not need a representation of S of the previous form, and can compute the generating series of such sets by hand. For instance, the generating series of \mathbb{N}^2 is $\sum_{n,m} x^n y^m = \frac{1}{(1-x)(1-y)}$, the generating series of $S_1 = \{(n, m) : n = m\}$ is $\sum_n x^n y^n = \frac{1}{1-xy}$, the generating series of $S_2 = \{(n, m) : n \neq m\} = \mathbb{N}^2 \setminus S_1$ is $\frac{1}{(1-x)(1-y)} - \frac{1}{1-xy}$. For a union, we can add the generating series, but we need to be careful to subtract the intersection, otherwise the vectors of the intersection would be counted twice. For instance, the generating series of $S_3 = \{(n, m, p) : n = m \text{ or } n = p\}$ is $\frac{1}{(1-xy)(1-z)} + \frac{1}{(1-yz)(1-x)} - \frac{1}{1-xyz}$.

3 Infinite ambiguity

Let L be a context-free language. For $n \in \mathbb{N}$, we recall that ℓ_n denotes the number of words in L of length n . In this section, we show how the asymptotic behaviour of ℓ_n can sometimes be sufficient to prove the inherent infinite ambiguity of L .

► **Definition 2** (finite degree of ambiguity). *Let $k \in \mathbb{N}$. A context-free grammar G is said to be k -ambiguous if every word $w \in \mathcal{L}(G)$ admits at most k different derivation trees. Similarly a context-free language L is k -ambiguous if it can be recognized by a k -ambiguous CFG.*

If such a finite k exists, then L is said to be of bounded ambiguity, or finitely ambiguous; otherwise, L is said to be of unbounded ambiguity, or infinitely ambiguous.

Infinitely ambiguous languages can arise from the concatenation of simple unambiguous languages; for instance, the language Pal of palindromes is unambiguous, but the language $Pal^2 = \{w_1 w_2 : w_1, w_2 \in Pal\}$ is infinitely ambiguous [7]. For infinitely ambiguous grammars, the functions $f(n)$ upper-bounding the number of different derivations of words of length n have been well studied [31, 32, 33]. Note that deciding infinite ambiguity is also undecidable [15]. The usual studies on finite or infinite ambiguity rely generally on iterations with Ogden’s lemma or Ullian and Ginsburg’s criteria (see for instance [29] which gives examples, for each $k \in \mathbb{N}$, of arbitrary inherently k -ambiguous on $a^* b^* c^*$). In this section we propose a novel approach based on generating series and their asymptotic behaviour.

3.1 An analytic criterion for infinite ambiguity

Let G be a context-free grammar such that every word $w \in \mathcal{L}(G)$ has a finite number of derivation. We call $G(x) = \sum_{n \in \mathbb{N}} g_n x^n$ the generating series of the derivation trees of G , where g_n denotes the number of derivation trees for words of $\mathcal{L}(G)$ of length n . Then, by the Chomsky-Schützenberger theorem [6], $G(x)$ is algebraic. More precisely, $G(x)$ belongs to a more restrictive class of algebraic series, called \mathbb{N} -algebraic series, for which the asymptotic behaviour of the coefficient has been well studied:

► **Proposition 3** (Critical exponents of \mathbb{N} -algebraic series [1]). *Let $G(z) = \sum_n g_n z^n$ be an \mathbb{N} -algebraic series. If G has a unique singularity on its circle of convergence $|z| = 1/\beta$, then*

$$g_n \sim_{n \rightarrow \infty} \frac{C}{\Gamma(1 + \alpha)} n^\alpha \beta^n, \tag{1}$$

where C, β are non negative algebraic constants, and α belongs to the following set:

$$\mathbb{D}_2 := \{-1 - 2^{-(k+1)} : k \geq 0\} \cup \left\{ -1 + \frac{r}{2^k} : k \geq 0, r \geq 1 \right\}.$$

If $G(z)$ has several dominant singularities, then there exists a non negative integer p such that for every $s \in [0, p - 1]$, either $g_{s+np} = 0$ for all n sufficiently large, or g_{s+np} has an asymptotic behaviour of the form of (1), where each constant depends on s .

We now derive the following criterion for infinite ambiguity, where we recall that \mathbb{D}_2 is defined in Proposition 3, and $n \equiv s[p]$ means that n is congruent to s modulo p :

► **Theorem 4.** *Suppose that it is not possible to find an integer $p \in \mathbb{N}_{>0}$ such that for all integer $s \in \{0, \dots, p-1\}$, for all $n \equiv s[p]$, $\ell_n = 0$ or ℓ_n satisfies a relation of the form $\ell_n = \Theta(\beta_s^n n^{\alpha_s})$ with $\beta_s > 0$ algebraic, and $\alpha \in \mathbb{D}_2$. Then L is infinitely ambiguous.*

Proof. We prove the contraposition: assume that L is k -ambiguous for some $k \in \mathbb{N}_{>0}$, and let us show that it is possible to find an integer $p > 0$ such that for all $s \in [0, p-1]$, for all $n \equiv s[p]$, $\ell_n = 0$ or ℓ_n satisfies a relation of the form $\ell_n = \Theta(\beta_s^n n^{\alpha_s})$ with $\beta_s > 0$ algebraic, and $\alpha \in \mathbb{D}_2$.

Let $L(x) = \sum_n \ell_n x^n$ the generating series of L , and $G(x) = \sum_n g_n x^n$ the generating series of the derivations of G . Then by definition of k -ambiguity, for every $n \in \mathbb{N}$, $\ell_n \leq g_n \leq k\ell_n$. In other words, $\frac{g_n}{k} \leq \ell_n \leq g_n$, which implies that $\ell_n = \Theta(g_n)$, where g_n is the coefficient of an \mathbb{N} -algebraic series. By Proposition 3, there exists a non negative integer p such that for every $s \in \{0, \dots, p-1\}$, either $g_{s+np} = 0$ for all n sufficiently large, or g_{s+np} has an asymptotic behaviour of the form of (1).

If $g_{s+np} = 0$ for all n sufficiently large, then so is ℓ_{s+np} . If $g_{s+np} \neq 0$ for n sufficiently large, then there exist C a constant, β_s a non negative algebraic number, and $\alpha_s \in \mathbb{D}_2$ such that $g_n \sim \frac{C}{\Gamma(1+\alpha_s)} n^{\alpha_s} \beta_s^n$ when $n \rightarrow \infty$ with $n \equiv s[p]$. Hence $\ell_n = \Theta(\beta_s^n n^{\alpha_s})$. ◀

► **Corollary 5.** *Let L be a context-free language such that, as $n \rightarrow +\infty$, $\ell_n = \Theta(\beta^n n^\alpha \log(n)^s)$. If β is not algebraic, or if $s \neq 0$, or if $\alpha \notin \mathbb{D}_2$, then L is inherently infinitely ambiguous.*

Proof. By hypothesis, there exist two constants $b_1, b_2 > 0$ such that for n large enough,

$$b_1 \beta^n n^\alpha \log(n)^s \leq \ell_n \leq b_2 \beta^n n^\alpha \log(n)^s.$$

In particular, for n sufficiently large, $\ell_n > 0$. Without loss of generality, we can modify its first terms, and suppose that $\ell_n > 0$ for every $n \in \mathbb{N}$. Let us prove that the hypotheses of Theorem 4 are satisfied in the case where β is not algebraic, or $s \neq 0$, or $\alpha \notin \mathbb{D}_2$.

As $\ell_n > 0$ for every $n \in \mathbb{N}$, suppose by contradiction that there exists an integer $p > 0$ such that for all $n \equiv 0[p]$, ℓ_n can be expressed as $\ell_n = \Theta(\beta_0^n n^{\alpha_0})$, with β_0 a non negative algebraic constant, and $\alpha_0 \in \mathbb{D}_2$. Hence there exists two constants $c_1, c_2 > 0$ such that, for every $n \equiv 0[p]$ sufficiently large, $c_1 \beta_0^n n^{\alpha_0} \leq \ell_n \leq c_2 \beta_0^n n^{\alpha_0}$, and combining the two inequalities:

$$0 < \frac{c_1}{b_2} \leq \left(\frac{\beta}{\beta_0} \right)^n n^{\alpha - \alpha_0} \log(n)^s \leq \frac{c_2}{b_1}.$$

By predominance of the growth of the exponential, if $\beta_0 \neq \beta$, the term in the middle either tends to 0 or $+\infty$ and cannot be bounded by two strictly positive constants. Hence if β is not algebraic, $\beta_0 \neq \beta$ and we obtain a contradiction, so that L is infinitely ambiguous by Theorem 4. Otherwise if β is algebraic, $\beta = \beta_0$ and for all n sufficiently large with $n \equiv 0[p]$:

$$0 < \frac{c_1}{b_2} \leq n^{\alpha - \alpha_0} \log(n)^s \leq \frac{c_2}{b_1}.$$

Similarly, the only way for $n^{\alpha - \alpha_0} \log(n)^s$ to be bounded by two strictly positive constants is to have both $\alpha = \alpha_0$ and $s = 0$, hence if $s \neq 0$ or $\alpha \notin \mathbb{D}_2$, we obtain a contradiction, so that L is infinitely ambiguous by Theorem 4. ◀

3.2 Application to Shamir's language

Let us illustrate the method given in the previous section on Shamir's language. Let $\Sigma = \{\#, a_1, \dots, a_k\}$ be an alphabet of $k + 1$ letters, with $k \geq 2$. We consider the extended Shamir language L_k defined by :

$$L_k = \{w \in \Sigma \mid w = s\#us^Rv \text{ with } s, u, v \in \{a_1, \dots, a_k\}^* \text{ and } s \neq \varepsilon\},$$

where the letter $\#$ serves only as a separator, and s^R denotes the mirror⁴ of s . This language is easily recognised by the *ambiguous* context-free grammar defined by the rules $S \rightarrow AB$ and $\{A \rightarrow aAa \mid a\#Ba, B \rightarrow aB \mid \varepsilon : a \in \Sigma \setminus \{\#\}\}$.

For $k = 2$, the language L_2 is one of the languages showed to be infinitely ambiguous by Shamir [30], using iterations on derivations similar to Ogden's lemma (the author actually shows the finer result that most words in the language of the form $s\#w$ have as many derivation trees as there are instances of s^R in w).

We propose here an analytic proof of the infinite ambiguity of the language L_k . In the following, ℓ_n denotes the number of words of L_k of length n . The whole proof relies on the following bounds:

► **Proposition 6.** *There exist constants $b_1, b_2 > 0$ such that for n sufficiently large,*

$$b_1 \log_k n \leq \frac{\ell_n}{k^{n-1}} \leq b_2 \log_k n.$$

In other words, $\ell_n = \Theta(k^{n-1} \log_k(n))$.

Applying Corollary 5 provides an analytic proof of the infinite ambiguity of Shamir's language:

► **Corollary 7.** *The Shamir language L_k is infinitely ambiguous.*

► **Remark 8.** In [10], the series of a weaker version of Shamir's language is shown to have infinitely many singularities. We could wonder if the number of singularities of the generating series of a language was correlated to its degree of ambiguity. This is not the case: Flajolet [10] gave examples of 2-ambiguous languages with an infinite number of singularities; on the other hand, the language L^* with $L = \{a^n b^m c^p : n = m \text{ or } n = p\}$ has a rational generating series, hence a finite number of singularities, but is infinitely ambiguous [24, Satz 4.2.1].

4 Two simple criteria on generating series for proving the inherent ambiguity of bounded languages

In this section, we revisit Ginsburg and Ullian's criteria with generating series. We develop simple methods to prove the inherent ambiguity of bounded languages without any iteration argument. Let us fix a dimension $d \geq 1$, and Σ an alphabet⁵ of cardinality more than 2.

4.1 Bounded languages and Ullian and Ginsburg's criteria

Let us fix a tuple of d words $w_1, \dots, w_d \in \Sigma^*$, denoted by $\langle w \rangle := \langle w_1, \dots, w_d \rangle$. We use the same notation and definition of [14]. A language L is called *bounded* with respect to $\langle w \rangle$ if $L \subseteq w_1^* \dots w_d^*$. The fonction $f_{\langle w \rangle} : \mathbb{N}^d \rightarrow w_1^* \dots w_d^*$ is defined by $f_{\langle w \rangle}(p_1, \dots, p_d) = w_1^{p_1} \dots w_d^{p_d}$

⁴ If $s = s_1 s_2 \dots s_{n-1} s_n$, then $s^R = s_n s_{n-1} \dots s_2 s_1$.

⁵ Context-free languages on an alphabet of size 1 are regular languages by Parikh theorem [27].

for every $\mathbf{p} \in \mathbb{N}^d$. Notice that if every w_i is a distinct letter of Σ , then the function $f_{\langle w \rangle}$ is bijective, and its inverse is the Parikh image on $w_1^* \dots w_d^*$. A bounded language L with respect to $\langle w \rangle$ is called *semilinear* if $f_{\langle w \rangle}^{-1}(L)$ is a semilinear set. By [14], every bounded context-free language is semilinear. In practice, most bounded languages are defined by giving explicitly their semilinear set $f_{\langle w \rangle}^{-1}(L)$. For instance, if $L = \{a^i b^j c^k : i = j \text{ or } j = k\}$, then $f_{\langle a,b,c \rangle}^{-1}(L) = \{(i, j, k) \in \mathbb{N}^3 : i = j \text{ or } j = k\}$.

The following definition introduces a crucial class of sets associated to bounded languages:

► **Definition 9** (Stratified set, [13, 14]). *A subset $X \subseteq \mathbb{N}^d$ is stratified if :*

1. *every element of X has at most two non-zero coordinates ;*
2. *it is not possible to find four integers $1 \leq i < j < k < m \leq d$ and two vectors $\mathbf{x}, \mathbf{x}' \in X$ such that $x_i x'_j x_k x'_m \neq 0$. In other words, two distinct elements of X cannot have “interlacing” nonzero coordinates.*

We sometimes say abusively that a linear set is stratified if its set of periods is stratified. Stratified sets of periods play a fundamental role in the form of the semilinear sets described by context-free grammars. In [13], Ginsburg and Ullian show that a bounded language L with respect to $a_1^* \dots a_d^*$, where $\langle a \rangle = \langle a_1, \dots, a_d \rangle$ are distinct letters, is context-free if and only if $f_{\langle a \rangle}^{-1}(L)$ is a finite union of linear sets, each with a stratified set of periods. They specialized this result for unambiguous bounded languages:

► **Theorem 10** (Ginsburg and Ullian criteria, [14]). *Let L be a context-free language bounded with respect to $\langle w \rangle = \langle w_1, \dots, w_d \rangle$. Then L is inherently ambiguous if and only if $f_{\langle w \rangle}^{-1}(L)$ is not a finite union of disjoint linear sets, each with a stratified set of periods whose vectors are linearly independent.*

► **Remark 11.** Note that it is not necessary, in order to use this criterion, to impose the decomposition of a word of L into $w_1^* \dots w_d^*$ to be unambiguous.

One direction of the equivalence can be easily understood in the case where every w_i are distinct symbols and the semilinear set associated to L is a disjoint union of linear sets with linearly independent stratified set of periods. One can easily build an unambiguous grammar recognising the language of each linear set (the non-interlacing condition makes it possible to order the vectors of the periods according to their non-zero pairs of coordinates, in a way that they are well nested). The other direction is the heart of Ginsburg and Ullian’s theorem, and is based on deep arguments⁶ about derivation trees.

As we mentioned it in the introduction, these criteria are powerful as they succeeded in leaving the world of grammars and derivation trees, to focus on the semilinear set behind the language. However, this characterisation of inherent ambiguity does not provide any tool to prove that a given semilinear set cannot be written as a finite union of disjoint stratified linear sets with independent periods. Hence, most proofs based on this result (see for instance [14, 16, 29]) mimicked on semilinear sets the iteration arguments that worked on derivation trees, without taking fully advantage of the fact that \mathbb{N}^d and its semilinear sets are much more amenable to techniques of analysis or algebra than derivation trees.

The next sections are devoted to show how Ginsburg and Ullian’s theorem actually translates nicely in the world of generating series, and thus allows to derive very simple criteria to prove the inherent ambiguity of many bounded languages.

⁶ As one of the authors admits it in his book [12, p. 188], “*The proof of the necessity is extremely complicated*”.

4.2 The three variables criterion

The theorem of this section is a simple criterion to prove the inherent ambiguity of bounded languages using generating series. The proof relies on the criteria of Ginsburg and Ullian, and some arithmetic in $K[\mathbf{x}]$, including the unicity of the decomposition into irreducible factors. Even if we only need $K = \mathbb{Q}$ to apply the theorem to the examples of this article, we state it in the general case where K is an arbitrary field. In particular, with $K = \mathbb{F}_2$, it generalises the criterion of [21], which only deals with bounded languages on distinct letters.

► **Theorem 12** (Three variables criterion). *Let $L \subseteq w_1^* \dots w_d^*$ be a context-free language bounded with respect to $\langle w \rangle$. Let $S = f_{\langle w \rangle}^{-1}(L)$ its associated semilinear set, and let*

$$S(x_1, \dots, x_d) = \frac{P(x_1, \dots, x_d)}{Q(x_1, \dots, x_d)} \in K(x_1, \dots, x_d)$$

be the generating series of S , such that P and Q are polynomials of $K[x_1, \dots, x_d]$ (that need not to be coprime). Suppose that there exists an irreducible polynomial $D \in K[x_1, \dots, x_d]$ that divides Q , does not divide P , and depends on more than three variables (in other words $D \notin K[x_i, x_j]$ for all $1 \leq i, j \leq d$). Then L is inherently ambiguous.

Proof. Suppose that L is unambiguous. By Ginsburg et Ullian's criteria (Theorem 10), the semilinear set S can be written under the form $S = \bigsqcup_{i=1}^r (\mathbf{c}_i + P_i^*)$, where the union is disjoint, each P_i is stratified, and the vectors in each set of periods P_i are linearly independent.

The disjoint union as well as the independent periods mean that this is an unambiguous description of S , such that its generating series is given by:

$$\frac{P(\mathbf{x})}{Q(\mathbf{x})} = S(\mathbf{x}) = \sum_{i=1}^r \frac{\mathbf{x}^{\mathbf{c}_i}}{\prod_{\mathbf{p} \in P_i} (1 - \mathbf{x}^{\mathbf{p}})} = \frac{P_2(\mathbf{x})}{Q_2(\mathbf{x})}, \text{ with } Q_2(\mathbf{x}) = \prod_{i=1}^r \prod_{\mathbf{p} \in P_i} (1 - \mathbf{x}^{\mathbf{p}}),$$

where P_2, Q_2 are obtained by writing the sum of fractions on the same denominator. Hence $PQ_2 = P_2Q$. The irreducible polynomial D divides Q , so it divides P_2Q , hence it divides PQ_2 ; as D is irreducible and does not divide P , it divides Q_2 .

However, as S is stratified, no period vector \mathbf{p} in any P_i has more than two non zero coordinates. This means that Q_2 is a product of polynomials of the form $(1 - t)$ where t is a monomial with at most two variables. Each of these polynomials admits a unique factorization in irreducible polynomials, each of them having at most two variables. By the unicity of the irreducible factorization in $K[x_1, \dots, x_d]$, D cannot divide Q_2 since it is irreducible with more than three variables. Contradiction. ◀

► **Remark 13.** As seen in the preliminaries, every semilinear set can be described unambiguously [8, 18], so that it is always possible to compute its generating series.

► **Proposition 14.** *The following context-free languages are inherently ambiguous:*

1. $L_1 = \{a^i b^j c^k \text{ with } i = j \text{ or } j = k\}$ and $L'_1 = \{a^i b a^j b a^k b \text{ with } i = j \text{ or } j = k\}$
2. $L_2 = \{a^i b^j c^k \text{ with } i \neq j \text{ or } j \neq k\}$ and $L'_2 = \{a^i b a^j b a^k b \text{ with } i \neq j \text{ or } j \neq k\}$
3. $L_3 = \{a^i b^j c^k \text{ with } i = j \text{ or } j \neq k\}$ and $L'_3 = \{a^i b a^j b a^k b \text{ with } i = j \text{ or } j \neq k\}$
4. $C := \{w_1 w_2 : w_1, w_2 \in \{a, b\}^* \text{ are palindromes}\}$

Proof. We apply Theorem 12 (with $K = \mathbb{Q}$) by exhibiting three-variables irreducible factors in the denominator of the generating series of the semilinear sets under irreducible form.

1. The generating series $S(a, b, c)$ of the semilinear set associated to L_1 is:

$$\frac{1}{(1-ab)(1-c)} + \frac{1}{(1-bc)(1-a)} - \frac{1}{1-abc} = \frac{1-3a^2b^2c^2+2a^2b^2c+2ab^2c^2+2a^2bc-ab^2c+2abc^2-a^2b+2abc-bc^2-ac}{(1-a)(1-bc)(1-c)(1-ab)(1-abc)}$$

The polynomial $1 - abc$ in the denominator is irreducible in $\mathbb{Q}[a, b, c]$, and has three variables. Furthermore, $1 - abc$ does not divide the numerator (it can be checked with a computer algebra software, or by hand: in $\mathbb{Q}[a, b][c]$, the numerator is of degree 2 in c , so if $1 - abc$ divided it, the numerator would be of the form $(1 - abc)(\lambda c + \mu)$ with $\lambda, \mu \in \mathbb{Q}[a, b]$, so that each monomial in c^2 in the numerator should have ab in factor, which is not the case of the monomial $-bc^2$). Hence L_1 is inherently ambiguous by Theorem 12. Notice that the generating series of the semilinear set associated to L'_1 is simply $b_1b_2b_3S(a_1, a_2, a_3)$ where b_1, b_2, b_3 are associated to the three letters b , and a_1, a_2, a_3 are associated to the groups of a 's. Hence L'_1 is also inherently ambiguous.

2. The associated generating series is $\frac{1}{(1-a)(1-b)(1-c)} - \frac{1}{1-abc} = \frac{a+b+c-ab-ac-bc}{(1-a)(1-b)(1-c)(1-abc)}$. The irreducible polynomial $1 - abc$ has three variables, and does not divide the numerator, since its total degree is 3, whereas the numerator is of total degree 2. Hence L_2 , and similarly L'_2 are inherently ambiguous.

► **Remark 15.** The languages L_1 and L_2 were already proved to be inherently ambiguous in [21] with the same argument. Our criterion makes it possible to extend the criterion on word-bounded languages, to prove that L'_1 and L'_2 are also inherently ambiguous.

3. The generating series of the semilinear set associated to L_3 is

$$\frac{1}{(1-a)(1-b)(1-c)} - \left(\frac{1}{(1-a)(1-bc)} - \frac{1}{1-abc} \right) = \frac{3ab^2c^2-2ab^2c-2abc^2-b^2c^2+b^2c+bc^2+ab+ac-2bc-a+1}{(1-a)(1-b)(1-c)(1-bc)(1-abc)}$$

and the proof is similar as before for both L_3 and L'_3 .

4. This example illustrates why criteria on bounded languages on words are more useful than on distinct letters. The language C is known to be infinitely ambiguous [7]. Let us propose a new elementary proof of just its inherent ambiguity. Suppose that C is unambiguous. Then $\tilde{C} := C \cap ba^+ba^+abbaa^+ba^+b$ would be unambiguous, by stability of unambiguous context-free languages under intersection with a regular language [14]. As

$$\tilde{C} = \{ba^nba^mbba^pba^qb : (n = q \wedge m = p) \text{ or } (n = m \wedge p = q), n, m, p, q \in \mathbb{N}_{>0}\}$$

is bounded with respect to $\langle b, a, b, a, b, a, b, a, b \rangle$, we associate the variables x, y, z, t to the a 's, and u_i for $i = 1 \dots 5$ for the five b 's. The generating series associated to $S' = \{(n, m, p, q) \in \mathbb{N}_{>0}^4 : (n = q \wedge m = p) \text{ or } (n = m \wedge p = q)\}$ is $S'(x, y, z, t) = xyz t \left(\frac{1}{(1-xt)(1-yz)} + \frac{1}{(1-xy)(1-zt)} - \frac{1}{1-xyz t} \right)$. Then the generating series associated to \tilde{C} is:

$$S(u_1, x, u_2, y, u_3, z, u_4, t, u_5) = u_1u_2u_3^2u_4u_5xyz t \frac{-3x^2z^2t^2y^2+2x^2zt^2y+2xz^2t^2y+2y^2tx^2z+2y^2txz^2\dots}{(1-xt)(1-yz)(1-xy)(1-zt)(1-xyz t)}$$

where we truncated the numerator due to lack of space. We can verify that the irreducible 4-variables polynomial $1 - xyz t$ does not divide the numerator, which proves that \tilde{C} is inherently ambiguous. Contradiction. So C is inherently ambiguous. ◀

- **Remark 16.** To check if a polynomial of the form $\pi = 1 - \mathbf{x}^{\mathbf{v}}$ with $\mathbf{v} \in (\mathbb{N}_{>0})^d$ does not divide the numerator P , we could also have introduced $d - 1$ new variables y_i , and perform the substitution $x_1 \leftarrow y_1^{-v_2}, x_d \leftarrow y_{d-1}^{v_{d-1}}$ and for $1 < i < d$, $x_i \leftarrow y_{i-1}^{v_{i-1}} y_i^{-v_{i+1}}$. After this substitution, π vanishes, so if after the substitution P is not the null fraction, then it is not divisible by π . This chained substitution aims specifically at cancelling π , and it is not difficult to show that if $\pi' = 1 - \mathbf{x}^{\mathbf{v}'}$ vanishes after the substitution, then \mathbf{v}' and \mathbf{v} are linearly dependent over \mathbb{Q} . This trick will be used with $d = 2$ for the second criterion of this article.

41:12 Techniques for Proving the Inherent Ambiguity of Context-Free Languages

The last language of the previous proposition shows that our criteria can also be useful to prove the inherent ambiguity of non bounded languages. Here we give an other example. The language of primitive words \mathcal{P} , defined formally by $\mathcal{P} = \{w \in \Sigma^* \mid \forall u \in \Sigma^*, w \in u^* \Rightarrow u = w\}$, is the set of words that are not the power of a smaller word. This language is challenging, as it is still an open question to know if it is context-free. In 1994, [28] showed that the generating series of \mathcal{P} is not algebraic, and hence that if it was context-free, then it would be inherently ambiguous. We propose a new proof of this fact.

► **Proposition 17** ([28]). *The language of primitive words is not an unambiguous context-free language.*

Proof. The language $\mathcal{P} \cap a^*ba^*ba^*b = \{a^nba^mba^pb : n \neq m \text{ or } m \neq p\} = L'_2$ is inherently ambiguous by Proposition 14. ◀

Related work A special case of Theorem 12 has already been proved by [21], in the case where each w_i is a distinct letter and $K = \mathbb{F}_2$, using completely different techniques: the author focused on $GF(2)$ grammars, a class of context-free grammars for which union is replaced by symmetric difference, and the concatenation of two languages K and L is replaced by a special concatenation $K \odot L$ which keeps only the words w of $K \cdot L$ which admit an odd number of decompositions of the form $w = w_k w_\ell$ with $w_k \in K$ and $w_\ell \in L$. In [21], the author studies the generating series associated to bounded languages in $a_1^* \dots a_d^*$ recognized by a $GF(2)$ grammar, and shows that the irreducible polynomials at their denominator can only have at most two variables. The author proves with this criterion the inherent ambiguity of the language $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$. At the end of the article, the author mentions Ginsburg and Ullian's criteria, saying that it would be possible to use them to prove the inherent ambiguity of the language L , but explains that the proof would not be simpler. We showed in this section that the equivalence of Ginsburg and Ullian actually translates directly into the criterion found by [21], while generalising it to bounded languages on words.

4.3 The interlacing criterion

The three variables criterion of Theorem 12 does not exploit the non interlacing condition of a stratified set. In particular, it fails on the language $L = \{a^n b^m a^p b^q \mid n = p \text{ or } m = q\}$, as the denominator of the series of its semilinear set is $(1 - ac)(1 - bd)(1 - a)(1 - b)(1 - c)(1 - d)$, which only contains irreducible polynomials of at most two variables. But $(1 - ac)(1 - bd)$ presents two irreducible polynomials with interlaced variables, hence it is natural to wonder if this could be a sign of inherent ambiguity. If so, we need however additional conditions, as such a pattern can also occur in unambiguous languages, such as in the language $\{a^n c^n : n \geq 0\} \cup \{b^n d^n : n \geq 0\}$ whose associated series is $\frac{1}{1-ac} + \frac{1}{1-bd} = \frac{2-ac-bd}{(1-ac)(1-bd)}$.

In this section, we establish a second criterion dealing with the interlacing condition (Theorem 21). We will use several technical lemmas: Lemmas 18 and 19 are classical algebra lemmas on polynomials, while Lemma 20 studies precisely the shape of irreducible polynomials dividing the denominators of series associated to stratified linear sets.

► **Lemma 18** (Irreducibility of $1 - x^n y^m$). *Let $n, m \in \mathbb{N}$. The polynomial $1 - x^n y^m$ is irreducible in $\mathbb{Q}[x, y]$ if and only if $n \wedge m = 1$.*

► **Lemma 19.** *Let $n, m \in \mathbb{N}_{>0}$. Then $1 - x^n y^m = (1 - x^\alpha y^\beta)P(x, y)$ where $\alpha \wedge \beta = 1$, and $P(x, y)$ is a non zero polynomial whose coefficients are in $\{0, 1\}$. Furthermore $\alpha = n/(n \wedge m)$ and $\beta = m/(n \wedge m)$.*

► **Lemma 20.** Let $S = \mathbf{c} + P^*$ a stratified linear set with linearly independent periods. Let $k \geq 1$, $n, m \geq 1$ be three integers such that $n \wedge m = 1$, and $i \neq j$ be two indices of variables, and y a fresh new variable. Then:

- if $(1 - x_i^n x_j^m)^k \mid \prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$, then $k = 1$;
- if $(1 - x_i^n x_j^m) \nmid \prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$, then $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})|_{x_i=y^m, x_j=y^{-n}} \neq 0$, seen as an element of $\mathbb{Q}(y)[\mathbf{x}]$, the ring of polynomials over the field $\mathbb{Q}(y)$.

The following theorem is our second criterion for proving the inherent ambiguity of bounded languages using the non-interlacing condition.

► **Theorem 21 (Interlacing criterion).** Let $L \subseteq w_1^* \dots w_d^*$ a context-free language bounded with respect to $\langle w \rangle$. Let us denote by $S = f_{\langle w \rangle}^{-1}(L)$ its semilinear set, and $S(x_1, \dots, x_d) = \frac{P(x_1, \dots, x_d)}{Q(x_1, \dots, x_d)} \in \mathbb{Q}[x_1, \dots, x_d]$ its generating series, with P and Q two polynomials, non necessarily coprime. Suppose that:

1. Q is divided by two non-univariate irreducible polynomials $D(x_j, x_\ell)$ and $\pi(x_i, x_k)$ with interlaced indices $j < \ell$ and $i < k$ (i.e. $i < j < k < \ell$ or $j < i < \ell < k$);
2. $\pi(x_i, x_k)$ is of the form $\pi(x_i, x_k) = (1 - x_i^n x_k^m)$, with $n, m \geq 1$ and $n \wedge m = 1$;
3. finally, $D \nmid P|_{x_i=y^m, x_k=y^{-n}}$ in $\mathbb{Q}(y)[\mathbf{x}]$, where y is a fresh new variable.

Then L is inherently ambiguous.

Proof. Toward a contradiction, suppose that L is unambiguous. By Theorem 10, S can be written under the form $S = \bigsqcup_{s=1}^r (\mathbf{c}_s + P_s^*)$, where the union is disjoint, the periods P_i are stratified, and the vectors in each P_i are linearly independent. Its generating series is then:

$$\frac{P(\mathbf{x})}{Q(\mathbf{x})} = S(\mathbf{x}) = \sum_{s=1}^r \frac{\mathbf{x}^{\mathbf{c}_s}}{\prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})}$$

By hypothesis, $P|_{x_i=y^m, x_k=y^{-n}} \neq 0$ (as D always divides 0), so $\pi(x_i, x_k)$ does not divide P , and D does not divide P (otherwise D would divide $P|_{x_i=y^m, x_k=y^{-n}}$ as it is not affected by the substitution). Hence both π and D are irreducible polynomials of Q , that stay in the denominator after writing the fraction $S(\mathbf{x})$ under irreducible form. Hence they divide the least common multiple of every $\prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})$. Let us write $Q = (1 - x_i^n x_k^m) D(x_j, x_\ell) \tilde{Q}(\mathbf{x})$. Note that by Lemma 20, no irreducible factor of $\tilde{Q}(\mathbf{x})$ that stays after writing P/Q under irreducible form cancels at $x_i = y^m, x_k = y^{-n}$. Hence if $\tilde{Q}(\mathbf{x})|_{x_i=y^m, x_k=y^{-n}} = 0$, this means that an irreducible factor common between \tilde{Q} and P cancels with the substitution, but this is not possible since $P|_{x_i=y^m, x_k=y^{-n}} \neq 0$. So $\tilde{Q}(\mathbf{x})|_{x_i=y^m, x_k=y^{-n}} \neq 0$.

Let us write I_1 the set of indices s such that $(1 - x_i^n x_k^m) \mid \prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})$, and I_2 its complement. For every $s \in I_1$, let us write $\prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}}) = (1 - x_i^n x_k^m) R_s(\mathbf{x})$. By Lemma 20, $R_s|_{x_i=y^m, x_k=y^{-n}} \neq 0$, and by the non interlacing condition, no irreducible factor of R_s is a polynomial in exactly both variables x_j, x_ℓ . Hence, no irreducible factor⁷ of $R_s|_{x_i=y^m, x_k=y^{-n}}$ in $\mathbb{Q}(y)[\mathbf{x}]$ is a polynomial in exactly both variables x_j and x_ℓ .

By multiplying everything by π , we obtain the following equality in $\mathbb{Q}(\mathbf{x})$:

$$\sum_{s \in I_1} \frac{\mathbf{x}^{\mathbf{c}_s}}{R_s(\mathbf{x})} + (1 - x_i^n x_k^m) \sum_{s \in I_2} \frac{\mathbf{x}^{\mathbf{c}_s}}{\prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})} = \frac{P(\mathbf{x})}{D(x_j, x_\ell) \tilde{Q}(\mathbf{x})}.$$

For every $s \in I_2$, $\prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})|_{x_i=y^m, x_k=y^{-n}} \neq 0$ since $\pi \nmid \prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})$, by Lemma 20. Consequently, for every $s \in I_2$, $\frac{\mathbf{x}^{\mathbf{c}_s}}{\prod_{\mathbf{p} \in P_s} (1 - \mathbf{x}^{\mathbf{p}})} \Big|_{x_i=y^m, x_k=y^{-n}}$ is a well defined rational fraction

⁷ As $\mathbb{Q}(y)$ is a field, $\mathbb{Q}(y)[\mathbf{x}]$ is also factorial.

on \mathbf{x} with coefficients in $\mathbb{Q}(y)$. Hence, by evaluating at $x_i = y^m, x_k = y^{-n}$, we obtain the following equality on $\mathbb{Q}(y)(\mathbf{x})$:

$$\sum_{s \in I_1} \frac{\mathbf{x}^{c_s} |_{x_i=y^m, x_k=y^{-n}}}{R_s(\mathbf{x}) |_{x_i=y^m, x_k=y^{-n}}} = \frac{P(\mathbf{x}) |_{x_i=y^m, x_k=y^{-n}}}{D(x_j, x_\ell) \tilde{Q}(\mathbf{x}) |_{x_i=y^m, x_k=y^{-n}}}.$$

As $D(x_j, x_\ell)$ has exactly two variables x_j and x_ℓ , it is unchanged by the substitution. Furthermore, it is easy to see that an irreducible polynomial of $\mathbb{Q}[\mathbf{x}]$ remains irreducible in $\mathbb{Q}(y)[\mathbf{x}]$. As $D \nmid P |_{x_i=y^m, x_k=y^{-n}}$, D stays an irreducible factor in $\mathbb{Q}(y)[\mathbf{x}]$ of the denominator of the fraction $\sum_{s \in I_1} \frac{\mathbf{x}^{c_s} |_{x_i=y^m, x_k=y^{-n}}}{R_s |_{x_i=y^m, x_k=y^{-n}}}$ once put under irreducible form.

However, none of the polynomials $R_s |_{x_i=y^m, x_k=y^{-n}}$ have irreducible factors that depend on both variables x_j, x_ℓ . We obtain a contradiction when reducing the sum on the same denominator. So L is inherently ambiguous. \blacktriangleleft

► **Remark 22.** The last condition can be in practice replaced by the weaker condition that there exists a rational number $\alpha \in \mathbb{Q}_{>0} \setminus \{1\}$ such that $D \nmid P |_{x_i=\alpha^m, x_k=\alpha^{-n}}$.

► **Remark 23.** If D is of the form $1 - x_j^p x_\ell^q$, by Remark 16 the last condition can be in practice replaced by the weaker condition that $P |_{x_i=y^m, x_j=z^q, x_k=y^{-n}, x_\ell=z^{-p}}$ is a non-null fraction, with y, z two fresh variables.

We now use the interlacing criterion to prove the following proposition:

► **Proposition 24.** *The following context-free languages are inherently ambiguous:*

1. $L_1 = \{a^i b^j c^k d^\ell : i = k \text{ or } j = \ell\}$
2. $L_2 = \{a^i b^j c^k d^\ell : i \neq k \text{ or } j \neq \ell\}$
3. $L_3 = \{a^i b^j c^k d^\ell : i = k \text{ or } j \neq \ell\}$ (and similarly $L_4 = \{a^i b^j c^k d^\ell : i \neq k \text{ or } j = \ell\}$)
4. $L'_2 = \{a^i b^j c^k d^\ell : 3i \neq 5k \text{ or } 2j \neq 3\ell\}$
5. $L_4 = \{a^i b^j c^k d^\ell : i < k \text{ or } i + j < k + \ell\}$

Proof. We illustrate in the proofs several ways of verifying the hypotheses of our criterion.

1. The generating series of the semilinear set is:

$$\frac{1}{(1-a)(1-b)(1-d)} + \frac{1}{(1-bd)(1-a)(1-c)} - \frac{1}{(1-ac)(1-bd)} = \frac{1-ab-ac-ad-bc-bd-cd+2abc+2abd+2acd+2bcd-3abcd}{(1-a)(1-bd)(1-a)(1-b)(1-c)(1-d)}$$

Then define $D(b, d) := 1 - bd$ and $\pi(a, c) := 1 - ac$, which are both irreducible and their variables are interlaced. Let P be the numerator $1 - ab - ac - ad - bc - bd - cd + 2abc + 2abd + 2acd + 2bcd - 3abcd$.

As $P|_{a=y, c=1/y} = \frac{2y^2bd-4ybd-y^2b-y^2d+2bd+2yb+2yd-b-d}{y}$, is of degree 1 in b , it is not divisible by $1 - bd$ in $\mathbb{Q}(y)[b, d]$. By Theorem 21, L_1 is inherently ambiguous.

2. The generating series of the semilinear set is:

$$\frac{1}{(1-a)(1-b)(1-c)(1-d)} - \frac{1}{(1-ac)(1-bd)} = \frac{abc+abd+acd+bcd-ab-2ac-ad-bc-2bd-cd+a+b+c+d}{(1-a)(1-bd)(1-a)(1-b)(1-c)(1-d)}$$

Still define $\pi := 1 - ac$, $D := 1 - bd$ and P be the numerator. As $(1 - bd) \nmid P|_{a=2, c=1/2} = \frac{1}{2}(bd - b - d - 1)$, L_2 is inherently ambiguous by Theorem 21 and Remark 22.

3. The generating series of the semilinear set is⁸:

$$\frac{1}{(1-a)(1-b)(1-c)(1-d)} - \frac{1}{1-bd} \left(\frac{1}{(1-a)(1-c)} - \frac{1}{(1-ac)} \right) = \frac{3abcd-2abc-2abd-2acd-bcd+ab+ac+ad+bc-bd+cd-a-c+1}{(1-a)(1-b)(1-c)(1-d)(1-bd)(1-ac)}$$

Still define $\pi = (1 - ac)$, $D = (1 - bd)$, and P be the numerator. For y, z two new variables, let us compute $P|_{a=y, b=z, c=y^{-1}, d=z^{-1}} = \frac{y^2z^2-2y^2z-2yz^2+y^2+4yz+z^2-2y-2z+1}{yz}$ which is a non null fraction. By Remark 23 and Theorem 21, L_3 is inherently ambiguous.

⁸ Because $i = k \vee j \neq \ell$ is equivalent to $\neg(\neg(i = k) \wedge j = \ell)$

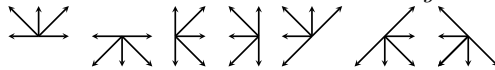
4. The associated generating series is $\frac{1}{(1-a)(1-b)(1-c)(1-d)} - \frac{1}{(1-b^3d^2)(1-a^5c^3)}$, that is:

$$S(a, b, c, d) = \frac{a^5b^3c^3d^2 - a^5c^3 - b^3d^2 - abcd + abc + abd + acd + bcd - ab - ac - ad - bc - bd - cd + a + b + c + d}{(1-a)(1-b)(1-c)(1-d)(1-b^3d^2)(1-a^5c^3)}$$
.
 Define $\pi = (1 - a^5c^3)$ and $D = (1 - b^3d^2)$, which are both irreducible with interlaced variables. Let P be the numerator. Let us choose $\alpha = 2$. As $(1 - b^3d^2) \nmid P|_{a=8, c=1/32} = \frac{217}{32}(bd - b - d + 1)$, L'_2 is inherently ambiguous⁹.
5. With a little more effort, we can check that the generating series associated to L_4 is $\frac{abcd^2 - acd - bd - cd + c + d}{(1-ac)(1-ad)(1-bd)(1-d)(1-c)(1-b)}$. Still define $D = 1 - bd$, $\pi = 1 - ac$ and P be the numerator. Let us choose $\alpha = 2$. As $P|_{a=2, c=1/2} = bd^2 - bd - d/2 + 1/2$ is not divisible by D , by Theorem 21 and Remark 22, L_4 is inherently ambiguous. ◀

► Remark 25. The previous proofs are based on the form of the semilinear set, and also work for their word-variant, like $\{a^i b a^j b a^k b a^\ell b : i \neq k \text{ or } j \neq \ell\}$.

4.4 An application to the complement of walks in the quarter plane

We consider the quarter plane \mathbb{N}^2 immersed in \mathbb{Z}^2 . We represent symbolically every vector of infinite norm 1 by an arrow symbol: \leftarrow represents $(-1, 0)$, \searrow represents $(1, -1)$, etc. The set of all these symbols $\mathcal{S} = \{\leftarrow, \swarrow, \downarrow, \searrow, \rightarrow, \nearrow, \uparrow, \nwarrow\}$ is called the set of *small steps* of \mathbb{Z}^2 . A word in \mathcal{S}^* can be represented by a walk in the plane, starting from $(0, 0)$, and following the vector represented by each letter. It is confined in the quadrant if every point of the path stays in the quarter plane \mathbb{N}^2 . For $\Sigma \subseteq \mathcal{S}$, we call W_Σ the language of words that are confined in the quarter plane. The study of such walks is an active domain of research in combinatorics (see for instance [3, 4, 5, 19, 22, 23]), as they provide a large diversity of generating series. Most of these walks are however not context-free languages, as there are two degrees of liberty. However, for every Σ , the language $\Sigma^* \setminus W_\Sigma$ of walks on Σ that leave the quadrant is context-free: a pushdown automaton non deterministically chooses one axis and accepts the word if the walk leaves this axis. A walk is called *singular* if Σ is a subset of one of the following sets¹⁰ [22]:



It is easy to see that singular walks are in fact unidimensional: their steps constrain the walk so that it cannot cross one of the two axes (except at $(0, 0)$), so that both W_Σ and $\Sigma^* \setminus W_\Sigma$ are easily unambiguous context-free [22]. On the contrary, with the two criteria of this section, we can prove the following proposition:

► **Proposition 26.** *The complement of every non-singular walk on the quarter plane is an inherently ambiguous context-free language.*

5 Conclusion

In conclusion, generating series are a beautiful and useful tool to study the question of inherent ambiguity on context-free languages. It would be interesting to find other criteria to study the inherent infinite ambiguity of languages that have simple asymptotic behaviour. Can we detect the infinite ambiguity of L^* , with $L = \{a^n b^m c^p : n = m \text{ or } n = p\}$ [24, Satz 4.2.1] using generating series? One lead would be to start by proving the inherent k -ambiguity of bounded languages, for a given k . For instance, if we can show that L^k is

⁹ We could also have checked that $P|_{a=y^3, b=z^2, c=y^{-5}, d=z^{-3}}$ is not the null fraction.
¹⁰ The last set is not called singular nor considered in [22], since walks with such steps leave the quadrant at the first step.

inherently $f(k)$ -ambiguous with $f(k) \rightarrow_{k \rightarrow \infty} \infty$ using generating series, then we could prove that L^* is inherently infinitely ambiguous. Ginsburg and Ullian's criteria (see [14, 29]) give a characterisation of the degree of ambiguity of bounded languages: a bounded context-free language is recognized by a k -ambiguous grammar if and only if its semilinear set can be decomposed as a finite union of stratified linear set, each with independent sets of periods, such that every intersection of $l > k$ of these linear sets is empty. This implies that the generating series of the semilinear set can be expressed by inclusion-exclusion as a sum of generating series of linear stratified sets and their Hadamard's product. It looks challenging to find a pattern that can only occur in the intersection of k stratified linear sets, or equivalently in the Hadamard's product of k of their generating series.

As for inherent ambiguity of bounded languages, finding inherently ambiguous languages that are not covered by Theorems 12 and 21 would be a nice challenge to improve them. We hope that having a stronger understanding on their series would help to determinate whether inherent ambiguity is decidable or not on bounded context-free languages.

References

- 1 Cyril Banderier and Michael Drmota. Formulae and asymptotics for coefficients of algebraic functions. *Combinatorics, Probability and Computing*, 24(1):1–53, 2015. doi:10.1017/S0963548314000728.
- 2 Jason P. Bell and Shaoshi Chen. Power series with coefficients from a finite set. *J. Comb. Theory, Ser. A*, 151:241–253, 2017. doi:10.1016/j.jcta.2017.05.002.
- 3 Alin Bostan, Frédéric Chyzak, Mark van Hoeij, Manuel Kauers, and Lucien Pech. Hypergeometric expressions for generating functions of walks with small steps in the quarter plane. *Eur. J. Comb.*, 61:242–275, 2017. doi:10.1016/j.ejc.2016.10.010.
- 4 Alin Bostan and Manuel Kauers. The complete generating function for Gessel walks is algebraic. *Proc. Amer. Math. Soc.*, 138(9):3063–3078, 2010. With an Appendix by Mark van Hoeij. doi:10.1090/S0002-9939-2010-10398-2.
- 5 Mireille Bousquet-Mélou and Marko Petkovsek. Walks confined in a quadrant are not always D-finite. *Theor. Comput. Sci.*, 307(2):257–276, 2003. doi:10.1016/S0304-3975(03)00219-6.
- 6 Noam Chomsky and Marcel-Paul Schützenberger. *The Algebraic Theory of Context-Free Languages*, volume 35 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1963. doi:10.1016/S0049-237X(08)72023-8.
- 7 JP Crestin. Un langage non ambigu dont le carré est d'ambiguïté non bornée. In *ICALP*, pages 377–390, 1972.
- 8 Samuel Eilenberg and Marcel-Paul Schützenberger. Rational sets in commutative monoids. *J. Algebra*, 13(2):173–191, 1969. doi:10.1016/0021-8693(69)90070-2.
- 9 Georges Elencwajg. Necessary and sufficient condition for $x^n - y^m$ to be irreducible in $[x, y]$. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/489703 (version: 2013-09-10). URL: https://math.stackexchange.com/q/489703.
- 10 Philippe Flajolet. Analytic models and ambiguity of context-free languages. *Theor. Comput. Sci.*, 49(2):283–309, 1987. doi:10.1016/0304-3975(87)90011-9.
- 11 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
- 12 Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., USA, 1966.
- 13 Seymour Ginsburg and Edwin Spanier. Semigroups, Presburger formulas, and languages. *Pac. J. Math.*, 16(2):285–296, 1966.
- 14 Seymour Ginsburg and Joseph Ullian. Ambiguity in context free languages. *J. ACM*, 13(1):62–89, 1966. doi:10.1145/321312.321318.

- 15 Sheila Greibach. A note on undecidable properties of formal languages. *Mathematical Systems Theory*, 2(1):1–6, 1968. doi:10.1007/BF01691341.
- 16 Thomas N. Hibbard and Joseph Ullian. The independence of inherent ambiguity from complementedness among context-free languages. *J. ACM*, 13(4):588–593, October 1966. doi:10.1145/321356.321366.
- 17 Juha Honkala. On parikh slender languages and power series. *Journal of Computer and System Sciences*, 52(1):185–190, 1996. doi:10.1006/jcss.1996.0014.
- 18 Ryuichi Ito. Every semilinear set is a finite union of disjoint linear sets. *J. Comput. Syst. Sci.*, 3(2):221–231, 1969. doi:10.1016/S0022-0000(69)80014-0.
- 19 Manuel Kauers and Alin Bostan. Automatic classification of restricted lattice walks. *Discrete Mathematics & Theoretical Computer Science*, 2009.
- 20 Serge Lang. *Algebra*, volume 211. Springer Science & Business Media, 2012.
- 21 Vladislav Makarov. Bounded languages described by $\text{gf}(2)$ -grammars. In Nelma Moreira and Rogério Reis, editors, *Developments in Language Theory – 25th International Conference, DLT 2021, Porto, Portugal, August 16-20, 2021, Proceedings*, volume 12811 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2021. doi:10.1007/978-3-030-81508-0_23.
- 22 Marni Mishna. Classifying lattice walks restricted to the quarter plane. *Journal of Combinatorial Theory, Series A*, 116(2):460–477, 2009.
- 23 Marni Mishna and Andrew Rechnitzer. Two non-holonomic lattice walks in the quarter plane. *Theoretical Computer Science*, 410(38-40):3616–3630, 2009.
- 24 Mohamed Najji. Grad der mehrdeutigkeit kontextfreier grammatiken und sprachen. Diplomarbeit, Johann Wolfgang Goethe-Universität, 1998.
- 25 William Ogden. A helpful result for proving inherent ambiguity. *Mathematical systems theory*, 2(3):191–194, 1968. doi:10.1007/BF01694004.
- 26 Rohit J Parikh. Language generating devices. *Quarterly Progress Report*, 60:199–212, 1961.
- 27 Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.
- 28 Holger Petersen. The ambiguity of primitive words. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 679–690. Springer, 1994.
- 29 Arnold L Rosenberg. A note on ambiguity of context-free languages and presentations of semilinear sets. *Journal of the ACM (JACM)*, 17(1):44–50, 1970.
- 30 Eliahu Shamir. Some inherently ambiguous context-free languages. *Inf. Cont.*, 18(4):355–363, 1971. doi:10.1016/S0019-9958(71)90455-4.
- 31 Klaus Wich. Exponential ambiguity of context-free grammars. In *Developments In Language Theory: Foundations, Applications, and Perspectives*, pages 125–138. World Scientific, 2000.
- 32 Klaus Wich. Sublinear ambiguity. In Mogens Nielsen and Branislav Rován, editors, *Mathematical Foundations of Computer Science 2000*, pages 690–698, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- 33 Klaus Wich. *Ambiguity functions of context-free grammars and languages*. PhD thesis, Universität Stuttgart, 2005.

A Proofs of Section 3

A.1 Proof of Proposition 6

► **Proposition 6.** *There exist constants $b_1, b_2 > 0$ such that for n sufficiently large,*

$$b_1 \log_k n \leq \frac{\ell_n}{k^{n-1}} \leq b_2 \log_k n.$$

In other words, $\ell_n = \Theta(k^{n-1} \log_k(n))$.

Proof. For a given prefix $s \in \Sigma^*$, we denote ℓ_n^s the number of words of size n in L_k of the form $s\#w$, and for $r \geq 1$, $\ell_n^r = \sum_{|s|=r-1} \ell_n^s$ denotes the number of words in L_k of the form $s\#w$, of length n , and such that $|s| = r - 1 \geq 1$.

Upper bound. Let us fix a word s , of length $r - 1$. We recall that ℓ_n^s counts the number of words of length n of the form $s\#w$, such that w contains the factor s^R . By removing this last constraint on w , we easily obtain that $\ell_n^s \leq k^{n-r}$.

Moreover, by partitioning according to the position j in the word w where the factor s^R appears, we also deduce $\ell_n^s \leq \sum_{j=0}^{n-2r+1} k^j k^{n-2r+1-j} = (n - 2r + 2)k^{n-2r+1} \leq nk^{n-2r+1}$.

Hence $\ell_n^s \leq \min(k^{n-r}, nk^{n-2r+1})$.

Note that $\min(k^{n-r}, nk^{n-2r+1}) = k^{n-r}$ if $r \leq \log_k n + 1$. Finally, for all $r \geq 2$, $\ell_n^r = \sum_{|s|=r-1} \ell_n^s \leq k^{r-1} \min(k^{n-r}, nk^{n-2r+1})$, and so we have:

$$\ell_n^r \leq \begin{cases} k^{n-1} & \text{if } r < \log_k n + 1 \\ nk^{n-r} & \text{if } r \geq \log_k n + 1 \end{cases}$$

Majoring ℓ_n by the sum of all ℓ_n^r , and partitioning according to the position of r with respect to $\log_k n + 1$, we obtain:

$$\begin{aligned} \ell_n &\leq \sum_{2 \leq r < \log_k n + 1} \ell_n^r + \sum_{r \geq \log_k n} \ell_n^r \leq k^{n-1}(\log_k n - 1) + nk^n \sum_{r \geq \log_k n + 1} k^{-r} \\ &\leq k^{n-1}(\log_k n - 1) + nk^n k^{-\log_k n - 1} \frac{k}{k-1} \\ &= k^{n-1}(\log_k n - 1) + k^{n-1} \frac{k}{k-1} \sim_{n \rightarrow \infty} k^{n-1} \log_k n \end{aligned}$$

So there exists a constant $b_2 > 0$ such that for n large enough, $\ell_n \leq b_2 k^{n-1} \log_k n$.

Lower bound. We still look at a word of size n of L_k of the form $s\#w$, with s of size $|s| = r - 1$. We split w into t consecutive blocks of size $r - 1$, with $t = \lfloor \frac{n-r}{r-1} \rfloor$. Note that the last remaining block is of size $r_1 = (n - r) - (r - 1)t$.

We want to lower-bound the number of words of L_k associated with a prefix s of size $r - 1$ by looking only at the words w having s^R as one of their t blocks. Thus:

$$\begin{aligned} \ell_n^s &\geq \text{card}\{w : s^R \text{ appears in one of the } t \text{ blocks of } w, \text{ with } |w| = n - r\} \\ &= k^{n-r} - \text{card}\{w : s^R \text{ does not appear in any of the } t \text{ blocks of } w, \text{ with } |w| = n - r\} \\ &= k^{n-r} - k^{r_1} (k^{r-1} - 1)^t = k^{n-r} \left(1 - \left(1 - \frac{1}{k^{r-1}} \right)^t \right) \end{aligned}$$

since $r_1 - (n - r) = -t(r - 1)$. As this bound depends only on $|s| = r$, we deduce that

$$\ell_n^r \geq k^{n-1} \left(1 - \left(1 - \frac{1}{k^{r-1}} \right)^t \right).$$

Notice that $\left(1 - \frac{1}{k^{r-1}} \right)^t = \exp\left(\lfloor \frac{n-r}{r-1} \rfloor \ln\left(1 - \frac{1}{k^{r-1}}\right)\right) \leq \exp\left(-\lfloor \frac{n-r}{r-1} \rfloor \frac{1}{k^{r-1}}\right)$.

Fix r such that $r \leq 1 + \frac{\log_k n}{2}$. Then $-\frac{1}{k^{r-1}} \leq -\frac{1}{\sqrt{n}}$. Besides, for $n \geq 4$, $n - r \geq \frac{n}{2}$ and $n - \log_k n \geq \frac{n}{2}$. Hence $\lfloor \frac{n-r}{r-1} \rfloor \geq \frac{n-r}{r-1} - 1 \geq \frac{n}{\log_k n} - 1 \geq \frac{n}{2 \log_k n}$. Consequently for $r \leq \frac{\log_k n}{2} + 1$:

$$1 - \left(1 - \frac{1}{k^{r-1}} \right)^t \geq (1 - \exp(-\frac{\sqrt{n}}{2 \log_k n})),$$

where the right side does not depend on r , and tends to 1 when $n \rightarrow \infty$. So for n sufficiently large, we can lower-bound it by $1/2$. Thus there exists a rank $n_0 > 0$ independent of r such that for every $n \geq n_0$ and $r \leq \frac{\log_k n}{2} + 1$, we have $\ell_n^r \geq \frac{1}{2} k^{n-1}$.

Hence, for $n \geq n_0$, $\ell_n \geq \sum_{r-1 \leq \frac{1}{2} \log_k n} \ell_n^r \geq \frac{1}{4} k^{n-1} \log_k n$. ◀

B

 Proofs of Section 4

B.1 Proof of Lemma 18

We need the following classical folklore lemma:

► **Lemma 27.** *If $f \in \mathbb{Q}[x, y]$ is homogenous, so is any of its divisors.*

Proof. Let us factorize $f = gh$, with $g, h \in \mathbb{Q}[x, y]$. We can decompose $g = \sum_{i=s}^r g_i$ and $h = \sum_{i=s'}^{r'} h_i$ as a sum of homogenous polynomials where for every i , h_i and g_i are either zero or of total degree i . Furthermore, let us suppose that $g_s, g_r, h_{s'}$ and $h_{r'}$ are non zero. Hence $f = (\sum_{i=s}^r g_i)(\sum_{i=s'}^{r'} h_i)$, and the highest total degree term of f is $g_r h_{r'}$, of total degree $r + r'$, and the lowest total degree term is $g_s h_{s'}$, of total degree $s + s'$. As f is homogenous, $r + r' = s + s'$, and as $s \leq r$ and $s' \leq r'$, we get that $s = r$ and $s' = r'$; this means that g and h are homogenous. ◀

The following lemma is also folklore, the proof is given for completeness.

► **Lemma 18 (Irreducibility of $1 - x^n y^m$).** *Let $n, m \in \mathbb{N}$. The polynomial $1 - x^n y^m$ is irreducible in $\mathbb{Q}[x, y]$ if and only if $n \wedge m = 1$.*

Proof. If n and m are not coprime, let $\delta > 1$ be a common divisor. Then $1 - x^n y^m = 1 - (x^{n/\delta} y^{m/\delta})^\delta = (1 - x^{n/\delta} y^{m/\delta}) \sum_{k=1}^{\delta-1} x^{kn/\delta} y^{km/\delta}$ is not irreducible.

If n and m are coprime, we adapt the nice proof of [9], by making it a little more elementary and thus a little less elegant. Let us write $f = 1 - x^n y^m$, and decompose $f = gh$.

Without loss of generality, $g = (a_0 + \dots + a_{r'} x^r y^{r'})$ with $a_0 \neq 0$, $a_{r'} \neq 0$, r' is the degree of g in the variable y , and r is the degree in x of the polynomial that is the coefficient of $y^{r'}$. Similarly, $h = (a_0^{-1} + \dots + -a_{r'}^{-1} x^s y^{s'})$ with $a_0 \neq 0$, $a_{r'} \neq 0$, s' the degree of h in the variable y , and s the degree in x of the coefficient of $y^{s'}$. Then $r' + s' = m$, and we can suppose without loss of generality that $r' \neq 0$.

The polynomial $Y^{nm} - X^{nm} = Y^{nm} f(X^m, Y^{-n}) = Y^{nr'} g(X^m, Y^{-n}) Y^{ns'} h(X^m, Y^{-n})$ is homogenous. As $Y^{nr'} g(X^m, Y^{-n})$ and $Y^{ns'} h(X^m, Y^{-n})$ are both polynomials in $K[X, Y]$, they are homogenous by Lemma 27.

Hence $a_0 Y^{nr'} + \dots + a_{r'} X^{mr}$ is homogenous, and $mr = nr'$. As m and n are coprime, m divides r' , and as $r' \neq 0$, $m \leq r'$, and consequently $m = r'$ and $s' = 0$. So $h(x, y)$ is a polynomial $\tilde{h}(x)$ in x only, but $Y^{ns'} h(X^m, Y^{-n}) = Y^{ns'} \tilde{h}(X^m)$ is homogenous, so $\tilde{h}(X^m)$ is homogenous too. As $a_0 \neq 0$, h is a constant. So f is irreducible. ◀

B.2 Proof of Lemma 19

► **Lemma 19.** *Let $n, m \in \mathbb{N}_{>0}$. Then $1 - x^n y^m = (1 - x^\alpha y^\beta) P(x, y)$ where $\alpha \wedge \beta = 1$, and $P(x, y)$ is a non zero polynomial whose coefficients are in $\{0, 1\}$. Furthermore $\alpha = n/(n \wedge m)$ and $\beta = m/(n \wedge m)$.*

Proof. Let us write $\delta = n \wedge m$. Then $1 - x^n y^m = (1 - x^{n/\delta} y^{m/\delta}) P(x, y)$, where $P(x, y) = \sum_{k=1}^{\delta-1} x^{kn/\delta} y^{km/\delta}$ is non zero polynomial whose coefficients are in $\{0, 1\}$. By definition of gcd, $(n/\delta) \wedge (m/\delta) = 1$. ◀

B.3 Proof of Lemma 20

► **Lemma 20.** *Let $S = c + P^*$ a stratified linear set with linearly independent periods. Let $k \geq 1$, $n, m \geq 1$ be three integers such that $n \wedge m = 1$, and $i \neq j$ be two indices of variables, and y a fresh new variable. Then:*

- if $(1 - x_i^n x_j^m)^k \mid \prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$, then $k = 1$;
- if $(1 - x_i^n x_j^m) \nmid \prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$, then $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})|_{x_i=y^m, x_j=y^{-n}} \neq 0$, seen as an element of $\mathbb{Q}(y)[\mathbf{x}]$, the ring of polynomials over the field $\mathbb{Q}(y)$.

Proof. As the period vectors are linearly independent, there exist at most two vectors $\mathbf{p}_1, \mathbf{p}_2 \in P$ such that $(1 - \mathbf{x}^{\mathbf{p}_1})$ and $(1 - \mathbf{x}^{\mathbf{p}_2})$ are in $\mathbb{Q}[x_i, x_j]$.

Let us write $(1 - \mathbf{x}^{\mathbf{p}_1}) = (1 - x_i^{n_1} x_j^{m_1}) = (1 - x_i^{n_1/d_1} x_j^{m_1/d_1}) P_1(x_i, x_j)$, with $n_1, m_1 \geq 1$, $P_1(x_i, x_j)$ which is a non zero polynomial with coefficients in $\{0, 1\}$, and $d_1 = n_1 \wedge m_1$. In particular, $P_1(1, 1) \neq 0$. Similarly, let us write $(1 - \mathbf{x}^{\mathbf{p}_2}) = (1 - x_i^{n_2} x_j^{m_2}) = (1 - x_i^{n_2/d_2} x_j^{m_2/d_2}) P_2(x_i, x_j)$ with the same conditions and notations.

As $P_1(1, 1)$ can not be zero, P_1 is not divisible by any polynomial of the form $(1 - \mathbf{x}^{\mathbf{p}})$ (and the same holds for P_2). Furthermore, the other factors in the denominator $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$ are not divisible by the irreducible polynomials $(1 - x_i^{n_1/d_1} x_j^{m_1/d_1})$ et $(1 - x_i^{n_2/d_2} x_j^{m_2/d_2})$, as they do not depend on simultaneously x_i and x_j .

Finally, $(n_1/d_1, m_1/d_1) \neq (n_2/d_2, m_2/d_2)$, as otherwise we would have $d_2 \mathbf{p}_1 = d_1 \mathbf{p}_2$, implying that \mathbf{p}_1 and \mathbf{p}_2 would be linearly dependent.

Hence, every irreducible polynomial of the form $(1 - x_i^n x_j^m)$ with $n \wedge m = 1$ has multiplicity at most 1 in the unique irreducible factorization of $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$. The first point is proved. Note that every other irreducible factor depending on both x_i and x_j are divisors of polynomials with coefficients in $\{0, 1\}$.

The second point comes from the following additional observations:

- a non null polynomial with coefficients in $\{0, 1\}$, and consequently its divisors, does not become the null fraction by replacing some of its variables by y^m or y^{-n} . Indeed, when we write the rational fraction after the substitution on irreducible form, the denominator is a power of y , and the numerator is a sum of polynomials with positive coefficients.
- for $s \notin \{i, j\}$, $1 - x_s^{p_s}$ stays the same after the substitution, while $(1 - x_i^{p_i})|_{x_i=y^m} = 1 - y^{m p_i}$ is a non null polynomial in y , and $(1 - x_j^{p_j})|_{x_j=y^{-n}} = \frac{y^{n p_j} - 1}{y^{n p_j}}$ is a non null element of $\mathbb{Q}(y)$.
- similarly a polynomial of the form $(1 - x_i^{p_i} x_j^{p_j})$ with $p_i, p_j \geq 1$ and $\{x_i, x_j\} \neq \{x_i, x_j\}$ does not vanish by replacing x_i by y^m and x_j by y^{-n} . For instance, for $s \notin \{i, j\}$, $(1 - x_j^{p_j} x_s^{p_s})|_{x_j=y^{-n}} = \frac{y^{n p_j} - x_s^{p_s}}{y^{n p_j}}$ is a non null fraction.

By the previous observations, the only irreducible factors of $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$ that risk canceling after the substitution $x_i = y^m, x_j = y^{-n}$ are of the form $(1 - x_i^{n_1} x_j^{n_2})$ with $n_1, n_2 \geq 1$, $n_1 \wedge n_2 = 1$ and $(n, m) \neq (n_1, n_2)$. Then, the substitution replaces such a polynomial with the fraction $(1 - y^{m n_1 - n n_2})$ in $\mathbb{Q}(y)$, which becomes null if and only if $m n_1 - n n_2 = 0$, if and only if $n = n_1$ et $m = n_2$ (as $n \wedge m = 1$ and $n_1 \wedge n_2 = 1$). Consequently, no irreducible factor of $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})$ becomes zero in $\mathbb{Q}(y)[\mathbf{x}]$ after the substitution, so $\prod_{\mathbf{p} \in P} (1 - \mathbf{x}^{\mathbf{p}})|_{x_i=y^m, x_j=y^{-n}}$ is a product of non-null polynomials in $\mathbb{Q}(y)[\mathbf{x}]$, hence is non null. ◀

B.4 Computation of the last series of Proposition 24

Let us explain how we computed the series of the semilinear set associated to $L_4 = \{a^i b^j c^k d^l : i < k \text{ or } i + j < k + l\}$. Let us notice that $i < k \text{ or } i + j < k + l \Leftrightarrow \neg(i \geq k \text{ and } i + j \geq k + l)$.

Hence $S(a, b, c, d) = \frac{1}{(1-a)(1-b)(1-c)(1-d)} - \sum_{n \geq p \text{ and } n+m \geq p+q} a^n b^m c^p d^q$. Let us write

$$S_2(a, b, c, d) = \sum_{n \geq p \text{ and } n+m \geq p+q} a^n b^m c^p d^q = \sum_{n=0}^{+\infty} a^n \sum_{p=0}^n c^p \sum_{m=0}^{+\infty} b^m \sum_{q=0}^{(n-p)+m} d^q.$$

Then

$$\begin{aligned}
S_2(a, b, c, d) &= \frac{1}{1-d} \sum_{n=0}^{+\infty} a^n \sum_{p=0}^n c^p \sum_{m=0}^{+\infty} b^m (1 - d^{n-p+m+1}) \\
&= \frac{1}{(1-b)(1-c)(1-d)} \sum_{n=0}^{+\infty} a^n (1 - c^{n+1}) - \frac{d}{(1-d)(1-bd)} \sum_{n=0}^{+\infty} (ad)^n \sum_{p=0}^n (c/d)^p \\
&= \frac{1}{(1-b)(1-c)(1-d)} \left(\frac{1}{1-a} - \frac{c}{1-ac} \right) - \frac{d}{(1-d)(1-c/d)} \left(\frac{1}{1-ad} - \frac{c/d}{1-ac} \right)
\end{aligned}$$

Hence, we obtain after simplification that $S(a, b, c, d) = \frac{abcd^2 - acd - bd - cd + c + d}{(1-ac)(1-ad)(1-bd)(1-d)(1-c)(1-b)}$.

B.5 Extra properties

► **Lemma 28** (Announced in Remark 16). *Let π be polynomial of the form $\pi = 1 - x_1^{v_1} \dots x_k^{v_k}$ with $v_1, \dots, v_d > 0$ and $v_{k+1} = \dots = v_d = 0$ (we can without loss of generality rename the variables). We introduce $k - 1$ new variables y_i , and perform the substitution $x_1 \leftarrow y_1^{-v_2}$, $x_d \leftarrow y_{k-1}^{v_{k-1}}$ and for $1 < i < k$, $x_i \leftarrow y_{i-1}^{v_{i-1}} y_i^{-v_{i+1}}$. Notice that after this substitution, π vanishes. Suppose that a polynomial of the form $\pi' = 1 - \mathbf{x}^{\mathbf{v}'}$ vanishes after the substitution. Then \mathbf{v}' and \mathbf{v} are linearly dependent over \mathbb{Q} .*

Proof. It is easy to see that in the case where π' has a non null degree in x_i for $i > k$, then it does not vanish after the substitution. Hence π' only depends on x_1, \dots, x_k , and $v'_{k+1} = \dots = v'_d = 0$. After performing the substitution on $\mathbf{x}^{\mathbf{v}'}$, we hence obtain:

$$F(\mathbf{y}) = \left(\frac{1}{y_1^{v_2}} \right)^{v'_1} \left(\frac{y_1^{v_1}}{y_2^{v_3}} \right)^{v'_2} \dots \left(\frac{y_{k-2}^{v_{k-2}}}{y_{k-1}^{v_k}} \right)^{v'_{k-1}} (y_{k-1}^{v_{k-1}})^{v'_k}$$

For π' to be zero, the valuation of every variable y_i must be zero in F . For $1 \leq i \leq k - 1$, we notice that the valuation of y_i is equal to $v_i v'_{i+1} - v_{i+1} v'_i$, hence using a determinant

notation, $\begin{vmatrix} v_i & v_{i+1} \\ v'_i & v'_{i+1} \end{vmatrix} = 0$. This means that for all $1 \leq i \leq k - 1$, there exists λ_i such that

$\begin{pmatrix} v_i \\ v'_i \end{pmatrix} = \lambda_i \begin{pmatrix} v_{i+1} \\ v'_{i+1} \end{pmatrix}$, with $\lambda_i \neq 0$ since all the v_i 's are non null. Hence every vector $\begin{pmatrix} v_i \\ v'_i \end{pmatrix}$ is colinear to $\begin{pmatrix} v_1 \\ v'_1 \end{pmatrix}$, hence the matrix $\begin{pmatrix} v_1 & \dots & v_d \\ v'_1 & \dots & v'_d \end{pmatrix}$ has rank 1: \mathbf{v} and \mathbf{v}' are linearly dependent on \mathbb{Q} . ◀

► **Lemma 29.** *If D is an irreducible polynomial of $\mathbb{Q}[\mathbf{x}]$, and y is a fresh new variable, then D is also an irreducible polynomial of $\mathbb{Q}(y)[\mathbf{x}]$.*

Proof. By contradiction suppose that $D = fg$, with f, g two non constant polynomials of $\mathbb{Q}(y)[\mathbf{x}]$. Each coefficient of f is a rational fraction of y , of the form $p(y)/q(y)$, for which both p and q has a finite set of roots in \mathbb{Q} – and the same holds for g . Hence we can find a rational number $\alpha \in \mathbb{Q}$ that is not among these roots; when evaluating the equality $D = fg$ at $y = \alpha$, then D stays the same, and f and g becomes non constant polynomials in $\mathbb{Q}[\mathbf{x}]$, contradicting the irreducibility of D . ◀

C Proof sketch of Proposition 26

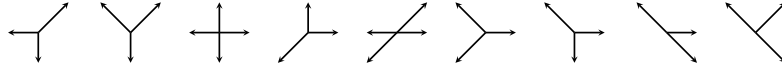
► **Proposition 26.** *The complement of every non-singular walk on the quarter plane is an inherently ambiguous context-free language.*

41:22 Techniques for Proving the Inherent Ambiguity of Context-Free Languages

Proof sketch. For $\Sigma \subseteq \mathcal{P} = \{\leftarrow, \rightarrow, \uparrow, \downarrow, \nearrow, \searrow, \nwarrow, \swarrow\}$, let us denote by $L_\Sigma = \Sigma^* \setminus W_\Sigma$ the language describing walks with steps in Σ that leave the quarter plane. Notice that if $\Sigma_1 \subseteq \Sigma_2 \subseteq \mathcal{P}$, as $L_{\Sigma_2} \cap \Sigma_1^* = L_{\Sigma_1}$, if L_{Σ_1} is inherently ambiguous, so is L_{Σ_2} .

Let us denote by σ the letter-morphism representing the axial symmetry of axis $y = x$ (for instance, $\sigma(\rightarrow) = \uparrow$, $\sigma(\nearrow) = \nearrow$, and $\sigma(\nwarrow) = \swarrow$). It is easy to see that for $\Sigma \subseteq \mathcal{P}$ and $w \in \Sigma^*$, $w \in L_\Sigma$ if and only if $\sigma(w) \in L_{\sigma(\Sigma)}$, so that L_Σ is inherently ambiguous if and only if $L_{\sigma(\Sigma)}$ is.

We then enumerate all non-singular sets $\Sigma \subseteq \mathcal{P}$, keep the minimal ones for inclusion, and choose arbitrarily one set per symmetry class with respect to σ . Then only nine sets remain to study:



We can divide those sets in three cases. In this sketched proof, we only detail the first case.

Case $\Sigma = \{\leftarrow, \downarrow, \nearrow\}$, $\Sigma = \{\nwarrow, \downarrow, \nearrow\}$ and $\Sigma = \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$

If $\Sigma = \{\leftarrow, \downarrow, \nearrow\}$, notice that $L_\Sigma \cap \nearrow^* \downarrow^* \leftarrow^* = \{\nearrow^n \downarrow^m \leftarrow^p : n < m \vee n < p\}$.

If $\Sigma = \{\nwarrow, \downarrow, \nearrow\}$, notice that $L_\Sigma \cap \nearrow^* \downarrow^* \nwarrow^* = \{\nearrow^n \downarrow^m \nwarrow^p : n < m \vee n < p\}$.

If $\Sigma = \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$, notice that $L_\Sigma \cap (\rightarrow)^* \downarrow^* \leftarrow^* = \{(\rightarrow)^n \downarrow^m \leftarrow^p : n < m \vee n < p\}$.

Let us call $S = \{(n, m, p) : n < m \vee n < p\}$. As $n < m \vee n < p \Leftrightarrow \neg(n \geq m \wedge n \geq p) \Leftrightarrow \neg(n \geq m \geq p \vee n \geq p > m)$, we have:

$$\begin{aligned} S(a, b, c) &= \frac{1}{(1-a)(1-b)(1-c)} - \frac{1}{(1-abc)(1-ab)(1-a)} - \frac{ac}{(1-abc)(1-ac)(1-a)} \\ &= \frac{a^2b^2c^2 - 2abc - cb + b + c}{(1-ac)(1-ab)(1-abc)(1-c)(1-b)} \end{aligned}$$

We can check that $1 - abc$ does not divide the numerator. Hence by Theorem 12, L_Σ is inherently ambiguous for every set Σ of this section.

The remaining two cases are similar: we can associate to $\Sigma = \{\rightarrow, \uparrow, \swarrow\}$, $\Sigma = \{\rightarrow, \nearrow, \swarrow, \nwarrow\}$ and $\Sigma = \{\rightarrow, \nwarrow, \swarrow\}$ the semilinear set $S = \{(n, m, p) : n < p \vee m < p\}$, of generating series $\frac{(1-ab)c}{(1-c)(1-a)(1-b)(1-abc)}$; and to $\Sigma = \{\downarrow, \rightarrow, \nwarrow\}$, $\Sigma = \{\swarrow, \rightarrow, \nwarrow\}$ and $\Sigma = \{\swarrow, \nearrow, \nwarrow\}$ the semilinear set $S = \{(n, m, p) : n < m \vee m < p\}$, of generating series $\frac{a^2b^2c - abc - ab - bc + b + c}{(1-abc)(1-a)(1-ab)(1-b)(1-c)}$. \blacktriangleleft

Synthesis of Privacy-Preserving Systems

Orna Kupferman  

The Hebrew University of Jerusalem, Israel

Ofer Leshkowitz  

The Hebrew University of Jerusalem, Israel

Abstract

Synthesis is the automated construction of a system from its specification. In many cases, we want to maintain the *privacy* of the system and the environment, thus limit the information that they share with each other or with an observer of the interaction. We introduce a framework for synthesis that addresses privacy in a simple yet powerful way. Our method is based on specification formalisms that include an *unknown* truth value. When the system and the environment interact, they may keep the truth values of some input and output signals private, which may cause the satisfaction value of specifications to become unknown. The input to the synthesis problem contains, in addition to the specification φ , also *secrets* ψ_1, \dots, ψ_k . During the interaction, the system directs the environment which input signals should stay private. The system then realizes the specification if in all interactions, the satisfaction value of the specification φ is true, whereas the satisfaction value of the secrets ψ_1, \dots, ψ_k is unknown. Thus, the specification is satisfied without the secrets being revealed. We describe our framework for specifications and secrets in LTL, and extend the framework also to the multi-valued specification formalism $LTL[\mathcal{F}]$, which enables the specification of the *quality* of computations. When both the specification and secrets are in $LTL[\mathcal{F}]$, one can trade-off the satisfaction value of the specification with the extent to which the satisfaction values of the secrets are revealed. We show that the complexity of the problem in all settings is 2EXPTIME-complete, thus it is not harder than synthesis with no privacy requirements.

2012 ACM Subject Classification Theory of computation; Theory of computation \rightarrow Automata over infinite objects

Keywords and phrases Synthesis, Privacy, LTL, Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.42

1 Introduction

Synthesis is the automated construction of a system from its specification: Given a linear temporal logic (LTL) formula φ over sets I and O of input and output signals, the goal is to return an I/O -transducer that *realizes* φ . At each moment in time, the transducer reads a truth assignment, generated by the environment, to the signals in I , and it generates a truth assignment to the signals in O . Thus, with every sequence of inputs, the transducer associates a sequence of outputs, and it realizes φ if all the computations that are generated by the interaction satisfy φ . Synthesis enables designers to focus on *what* the system should do rather than on *how* it should do it, and has attracted a lot of research and interest [41, 9].

While synthesized systems are correct, there is no guarantee about their quality. This is a real obstacle, as designers will give up manual design only after being convinced that the automatic process replacing it generates systems of comparable quality. An important quality measure is *privacy*: we seek systems that allow the underlying components not to reveal information they prefer to keep private. Unlike quality measures that are based on prioritizing different on-going behaviors, privacy is a global conceptual requirement, and it is not clear how to address the challenge of privacy in existing formulations and algorithms of synthesis. The Computer Science community has adopted the notion of *differential privacy* for formalizing when an algorithm maintains privacy. Essentially, an algorithm is differentially



© Orna Kupferman and Ofer Leshkowitz;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 42; pp. 42:1–42:23



Leibniz International Proceedings in Informatics

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

private if by observing its output, we cannot tell if a particular individual’s information is used in the computation [20, 22]. An orthogonal related challenge is that of *obfuscation* in system development, where we aim to develop systems whose internal operation is hidden [5, 27].

We introduce a framework for synthesis that addresses privacy in a simple yet powerful way. Our method is based on extending the semantics of the specification formalism to include an *unknown* truth value, denoted “?”. Let us first explain the framework when applied to LTL. In our framework, the input and output signals take values from $\{\mathbf{F}, ?, \mathbf{T}\}$, and so the semantics is defined with respect to an infinite *noisy computation* $\kappa \in (\{\mathbf{F}, ?, \mathbf{T}\}^{I \cup O})^\omega$. The satisfaction value of a formula ψ in κ is \mathbf{T} if all the computations obtained by “filling” the missing information in κ satisfy ψ , is \mathbf{F} if all these computations do not satisfy ψ , and is $?$ otherwise. Allowing the system and the environment to hide the values of some signals by assigning them $?$ is a natural way to increase privacy. Indeed, the truth value of these signals remains private. Adjusting the definition of realizability to require the system to satisfy the specification in all possible “fillings” of the missing values is also natural. Indeed, as is the case in other settings with incomplete information, we want the specification to hold no matter what the hidden values are [30].

An important question in the three-valued approach is how to measure the privacy level of the system. Clearly, the more signals are hidden, the more privacy is maintained. Still, an approach that is based on the number or the density of unknown assignments is not satisfactory, as many values are often not interesting, and we need an approach that captures the fact that the system and the environment do not reveal information they care not to reveal. A three-valued semantics for LTL and other temporal logics has already been studied in formal methods, in settings in which information is lost due to abstraction and other methods for coping with the state-explosion problem [10, 39, 18]. In all these methods, an evaluation of a specification to $?$ is a problem, which one should address by adding information, for example by refinement or revealing of data. The novelty of our approach is that we let the system and environment specify in LTL the behaviors they want to keep secret, and we view an evaluation to $?$ as a desirable outcome – when it applies to secret behaviors.

More formally, the input to our synthesis problem includes a specification φ and a list of *secrets* ψ_1, \dots, ψ_k , both over the set $I \cup O$ of input and output signals. The output of the synthesis problem is an *I/O-transducer with masking*. Such a transducer outputs, in each state, both a three-valued assignment to the signals in O (that is, some output signals may be assigned $?$), and a subset of input signals – these whose truth value should not be revealed in the current state. If in state s , the transducer asks the environment not to reveal the truth value of the signals in $M \subseteq I$, then the transitions from s are independent of the values of the signals in M . Accordingly, the interaction of a transducer \mathcal{T} with an environment produces an infinite noisy computation in $(\{\mathbf{F}, ?, \mathbf{T}\}^{I \cup O})^\omega$. The transducer is correct if for all input sequences $w_I \in (\{\mathbf{F}, \mathbf{T}\}^I)^\omega$, the interaction of \mathcal{T} with an environment that generates w_I result in a noisy computation $\kappa \in (\{\mathbf{F}, ?, \mathbf{T}\}^{I \cup O})^\omega$ such that the satisfaction value of the specification φ in κ is \mathbf{T} , and the satisfaction value of all secrets ψ_1, \dots, ψ_k in κ is $?$. In other words, all the computations of \mathcal{T} satisfy φ without revealing the secrets. Note that while in traditional synthesis, the system only decides what the assignment to the signals in O is, here the system decides which signals in $I \cup O$ it masks, and then decides what the assignment to the unmasked signals in O is. Clearly, the more signals the system masks, the easier it is for ψ_1, \dots, ψ_k to stay secret, and the harder it is for φ to be satisfied.

Recall that our goal of synthesizing systems that preserve privacy is a component in our overarching objective to synthesize systems of high quality. In recent years, researchers have started to address the challenge of synthesis of high-quality systems by extending the

Boolean setting to a multi-valued one, capturing different levels of satisfaction [8, 13, 1, 2]. We consider here the linear temporal logic $LTL[\mathcal{F}]$, which extends LTL with an arbitrary set \mathcal{F} of functions over $[0, 1]$. The satisfaction value of an $LTL[\mathcal{F}]$ formula φ is a value in $[0, 1]$, where the higher the satisfaction value is, the higher is the quality in which φ is satisfied [1]. Using the functions in \mathcal{F} , a specifier can prioritize different ways of satisfaction. Classical decision problems in the Boolean setting become optimization problems in the quantitative setting. In particular, in the synthesis problem, we seek systems with the highest possible satisfaction value [1, 2].

Adding privacy to the setting, this highest possible satisfaction value for the specification φ should be achieved without revealing the satisfaction value of the secrets. We follow the worst-case approach, where the quality of the synthesized system is the minimal satisfaction value of φ in some interaction, and the satisfaction value of all the secrets should be kept unknown in all interactions. We focus on secrets in LTL, but study also secrets in $LTL[\mathcal{F}]$, where we can also trade-off the satisfaction value of φ and the extent to which the satisfaction value of the secrets is revealed. We show that the complexity of the problem in all settings is 2EXPTIME-complete for specifications in LTL and $LTL[\mathcal{F}]$, thus it is not harder than synthesis with no privacy requirements.

As an example, consider a system that directs a robot patrolling a warehouse storage. Typical specifications for the system requires it to direct the robot so that it eventually reaches the shelves of requested items, it never runs out of energy, etc. Our algorithm automatically synthesizes a system that not only satisfies the specification, but also decides which parts of the interaction to hide so that the specification is satisfied without revealing secrets that would have been revealed by an observer of the full interaction. Such secrets may be dependencies between customers and shelves visited, locations of battery docking stations, and other properties of the structure of the warehouse. As a more specific example, assume there is a set of shelves $S = \{s_1, s_2, \dots, s_k\}$ such that we want to keep private the vicinity of shelves in S to docking stations. The input signals, namely these assigned by the robot, include the signals at_s_i , for $1 \leq i \leq k$, indicating the robot is at shelf s_i , and the signal *charging*, indicating the robot is at a docking station. Let $at_S = \bigvee_{i \in [k]} at_s_i$. Then, adding a secret $F(charging \wedge at_S)$ requires the system to direct the robot to hide the values of signals in a way that hides from an observer the truth value of “eventually, the robot is near both some shelf in S and a docking station”. Similarly, the secret $F(charging \wedge at_s_i)$ hides whether shelf s_i is near a docking station (recall that our framework supports a set of secrets, in particular a secret for each shelf in S). If we need to keep the whole radius of the charging docks secret, we can strengthen the secrets to $F(Xcharging \wedge (at_S \vee Xat_S \vee XXat_S))$, and similarly for the individual shelves. In order to prevent this secret from being evaluated to T, the system needs to direct the robot to assign ? to at_s_i not only when it assigns T to *charging*, but also in three time units around it, namely one time unit before and after making *charging* visible. Alternatively, the system can direct the robot to assign ? to *charging*. In addition, since the secret is evaluated to F in computations in which $at_s_i \wedge charging$ is always evaluated to F, the system needs to direct the robot to assign ? to at_s_i and *charging* in a way that prevents such an evaluation, for example by assigning them both ? in the initial state. In general, the choice of the system which signals to hide depends on other specifications it has to satisfy. If, for example, it is essential for the system to know about visits to all the shelves in S , then it may direct the robot not to charge near them or to hide the fact it does so. Otherwise, the system may leave the information about the visits unknown, and it may also combine the two solutions – this is exactly what our procedure does automatically.

One technical challenge of our algorithms is the need to combine automata for the specification with automata for the secrets. For the specification φ , the quantification of the hidden information is universal – we want all computations obtained by filling the hidden information to satisfy φ . For a secret ψ_i , the quantification of the hidden information is existential – we want witnesses that different fillings lead to different satisfaction values of ψ_i . The fact we need automata that handle both types of quantification makes it impossible to proceed with a *Safrless* synthesis algorithm, which requires universal automata [31]. We introduce and study a *syntax-based three-valued semantics* for LTL in noisy computations, which enables us to construct universal automata for secrets, leading to a Safrless synthesis algorithm that circumvents determinization and solution of parity games.

Related work. A very basic model of privacy has been studied in the context of synthesis with *incomplete information* [35, 30, 14], where the value of a subset of the signals stays secret throughout the interaction. Synthesis with incomplete information can be viewed as a special case of our approach here. Indeed, hiding of a signal p can be achieved with the secrets Fp and $F\neg p$. Moreover, our framework supports hiding of designated signals in parts (rather than all) of the interaction.

Lifting differential privacy to formal methods, researchers have introduced the temporal logic *HyperLTL*, which extends LTL with explicit trace quantification [17]. In particular, such a quantification can relate computations that differ only in non-observable elements, and can be used for specifying that computations with the same observable input have the same observable output. The synthesis problem of HyperLTL is undecidable, yet is decidable for the fragment with a single existential quantifier, which can specify interesting properties [24]. Our approach here is different, as it enables the specification of arbitrary secrets, and can be implemented on top of LTL synthesis tools.

As for obfuscation, while it is mainly studied in the context of software, where it has exciting connections with cryptography [5, 27], researchers have also studied the synthesis of obfuscation policies for temporal specifications [21, 43], which is closer to our approach here. In [43], an obfuscation mechanism is based on edit functions that alter the output of the system, aiming to make it impossible for an observer to distinguish between secret and non-secret behaviors. In [21], the goal is to synthesize a control function that directs the user which actions to disable, so that the observed sequence of actions would not disclose a secret behavior. Our work, on the other hand, addresses the general synthesis problem and thus handles specifications and secrets that are on-going infinite behaviors given by LTL and LTL[\mathcal{F}] specifications. In particular, while our transducers can mask information, they do not have an option to edit the interaction or disable actions of the environment.

2 Preliminaries

2.1 Automata

For a finite nonempty alphabet Σ , an infinite *word* $w = \sigma_1 \cdot \sigma_2 \cdot \dots \in \Sigma^\omega$ is an infinite sequence of letters from Σ . A *language* $L \subseteq \Sigma^\omega$ is a set of infinite words.

An *automaton* over infinite words is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Σ is an alphabet, Q is a finite set of *states*, $q_0 \in Q$ is an *initial state*, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a *transition function*, and α is an *acceptance condition*, to be defined below. For states $q, s \in Q$ and a letter $\sigma \in \Sigma$, we say that s is a σ -successor of q if $s \in \delta(q, \sigma)$. We consider automata with a total transition function. That is, for every state $q \in Q$ and letter $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| \geq 1$. If $|\delta(q, \sigma)| = 1$ for every state $q \in Q$ and letter $\sigma \in \Sigma$, then \mathcal{A} is *deterministic*.

A run of \mathcal{A} on $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states $r = r_0, r_1, r_2, \dots \in Q^\omega$, such that $r_0 = q_0$, and for all $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, \sigma_{i+1})$. The acceptance condition α determines which runs are “good”. We consider here the *Büchi*, *co-Büchi*, *generalized Büchi*, *generalized co-Büchi*, and *parity* acceptance conditions. All conditions refer to the set $\text{inf}(r) \subseteq Q$ of states that r traverses infinitely often. Formally, $\text{inf}(r) = \{q \in Q : q = r_i \text{ for infinitely many } i\}$. In generalized Büchi and co-Büchi automata, the acceptance condition is of the form $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_k\}$, for sets $\alpha_i \subseteq Q$. In a generalized Büchi automaton, a run r is accepting if for all $1 \leq i \leq k$, we have that $\text{inf}(r) \cap \alpha_i \neq \emptyset$. Thus, r visits each of the sets in α infinitely often. Dually, in a generalized co-Büchi automaton, a run r is accepting if there exists $1 \leq i \leq k$ such that $\text{inf}(r) \cap \alpha_i = \emptyset$. Thus, r visits at least one of the sets in α only finitely often. Büchi and co-Büchi automata are special cases, with $k = 1$, of their generalized forms. Finally, in a parity automaton $\alpha : Q \rightarrow \{1, \dots, k\}$ maps states to ranks, and a run r is accepting if the maximal rank of a state in $\text{inf}(r)$ is even. Formally, $\max_{q \in \text{inf}(r)} \{\alpha(q)\}$ is even. A run that is not accepting is *rejecting*. We refer to the number k in α (that is, the number of sets in the generalized conditions and the number of ranks in parity conditions) as the *index* of the automaton.

Note that as \mathcal{A} may not be deterministic, it may have several runs on a word. We distinguish between two branching modes. If \mathcal{A} is a *nondeterministic* automaton, then a word w is accepted by \mathcal{A} if there is an accepting run of \mathcal{A} on w . If \mathcal{A} is a *universal* automaton, then a word w is accepted by \mathcal{A} if all the runs of \mathcal{A} on w are accepting. The language of \mathcal{A} , denoted $L(\mathcal{A})$, is the set of words that \mathcal{A} accepts. Two automata are *equivalent* if their languages are equivalent.

We denote the different classes of automata by three-letter acronyms in $\{D, N, U\} \times \{B, C, GB, GC, P\} \times \{W, T\}$. The first letter stands for the branching mode of the automaton (deterministic, nondeterministic, or universal); the second for the acceptance condition type (Büchi, co-Büchi, generalized Büchi, generalized co-Büchi, or parity); and the third indicates we consider automata on words or trees (in Appendix A, we define tree automata). For example, NBWs are nondeterministic Büchi word automata.

2.2 Parity games

A parity game is $\mathcal{G} = \langle V, E, v_0, \alpha \rangle$ is played between two players SYS and ENV. The set V of positions is partitioned into two disjoint sets $V = V_{\text{ENV}} \cup V_{\text{SYS}}$, controlled by SYS and ENV. Then, $E \subseteq (V_{\text{SYS}} \times V_{\text{ENV}}) \cup (V_{\text{ENV}} \times V_{\text{SYS}})$ is a transition relation, which we assume to alternate between $V_s \text{Sys}$ and $V_s \text{env}$, $v_0 \in V$ is an initial position, and $\alpha : V \rightarrow \{1, \dots, k\}$ is a parity winning condition.

A strategy $f_{\text{Sys}} : V^* \cdot V_{\text{SYS}} \rightarrow V_{\text{ENV}}$ for *Sys* maps a finite path in \mathcal{G} that ends in a position $u \in V_{\text{SYS}}$ to a next position $v \in V_{\text{ENV}}$ such that $(u, v) \in E$, and similarly for a strategy $f_{\text{Env}} : V^* \cdot V_{\text{ENV}} \rightarrow V_{\text{SYS}}$ for *Env*. When the two players play according to their strategies f_{Sys} and f_{Env} , the *outcome* of the game, denoted $\text{outcome}(f_{\text{SYS}}, f_{\text{ENV}})$, is the unique infinite path $\rho = v_0, u_0, v_1, u_1, \dots \in (V_{\text{SYS}} \cdot V_{\text{ENV}})^\omega$, where $v_0 \in V_{\text{SYS}}$ is the initial position, and for all $j \geq 0$, we have that $u_j = f_{\text{Sys}}(v_0, u_0, \dots, v_j)$ and $v_j = f_{\text{Env}}(v_0, u_0, \dots, v_{j-1}, u_{j-1})$.

A strategy f_{Sys} of SYS is *winning* if for every strategy f_{Env} for ENV from v_0 , we have that $\text{outcome}(f_{\text{SYS}}, f_{\text{ENV}})$ satisfies the winning condition α . We say that SYS wins \mathcal{G} , if it has a winning strategy.

2.3 The temporal logic LTL[\mathcal{F}]

The logic LTL[\mathcal{F}] is a multi-valued logic that extends the linear temporal logic LTL with an arbitrary set of functions $\mathcal{F} \subseteq \{f : [0, 1]^k \rightarrow [0, 1] : k \in \mathbb{N}\}$ called quality operators. For example, \mathcal{F} may contain the maximum or minimum between the satisfaction values of subformulas, their product, and their average. This enables the specifier to refine the Boolean correctness notion and associate different possible ways of satisfaction with different truth values [1].

Let AP be a finite set of Boolean atomic propositions. The syntax of LTL[\mathcal{F}] is given by the following grammar, where the symbol \mathbf{T} stands for True, p ranges over a set AP of atomic propositions, $\varphi_1, \varphi_2, \dots, \varphi_k$ are LTL[\mathcal{F}] formulas and $f : [0, 1]^k \rightarrow [0, 1] \in \mathcal{F}$.

$$\varphi := \mathbf{T} \mid p \mid f(\varphi_1, \varphi_2, \dots, \varphi_k) \mid \mathbf{X}\varphi_1 \mid \varphi_1 \mathbf{U} \varphi_2.$$

The *length* of φ , denoted $|\varphi|$, is the number of nodes in the generating tree of φ . Note that $|\varphi|$ bounds the number of sub-formulas of φ . The semantics of LTL[\mathcal{F}] is defined with respect to *computations* over AP . Let the calligraphic digit 2 denote the set $\{\mathbf{F}, \mathbf{T}\}$, where \mathbf{F} stands for False and \mathbf{T} stands for True. Given a computation $\pi = \pi_0, \pi_1, \dots \in (2^{AP})^\omega$ and a position $j \geq 0$, we use π^j to denote the suffix $\pi_j, \pi_{j+1}, \dots \in (2^{AP})^\omega$ of π . The semantics maps a computation $\pi \in (2^{AP})^\omega$ and an LTL[\mathcal{F}] formula φ to the *satisfaction value of φ in π* , denoted $\llbracket \pi, \varphi \rrbracket$. The satisfaction value is in $[0, 1]$, and is defined inductively as follows.

- $\llbracket \pi, \mathbf{T} \rrbracket = 1$.
- $\llbracket \pi, p \rrbracket = 1$ if $p \in \pi_0$ and $\llbracket \pi, p \rrbracket = 0$ if $p \notin \pi_0$.
- $\llbracket \pi, f(\varphi_1, \varphi_2, \dots, \varphi_k) \rrbracket = f(\llbracket \pi, \varphi_1 \rrbracket, \llbracket \pi, \varphi_2 \rrbracket, \dots, \llbracket \pi, \varphi_k \rrbracket)$.
- $\llbracket \pi, \mathbf{X}\varphi_1 \rrbracket = \llbracket \pi^1, \varphi_1 \rrbracket$.
- $\llbracket \pi, \varphi_1 \mathbf{U} \varphi_2 \rrbracket = \max_{i \geq 0} \{ \min(\llbracket \pi^i, \varphi_2 \rrbracket, \min_{0 \leq j < i} \llbracket \pi^j, \varphi_1 \rrbracket) \}$.

The logic LTL coincides with the logic LTL[\mathcal{F}] for $\mathcal{F} = \{\neg, \vee, \wedge\}$, which corresponds to the usual Boolean operators. Formally, for $x, y \in [0, 1]$, we have $\neg x = 1 - x$, $x \vee y = \max\{x, y\}$, and $x \wedge y = \min\{x, y\}$. To see that LTL indeed coincides with LTL[\mathcal{F}], note that for this \mathcal{F} , all formulas are mapped to $\{0, 1\}$ in a way that agrees with the semantics of LTL. When φ is an LTL formula, we say that a computation π *satisfies* φ , denoted $\pi \models \varphi$, iff $\llbracket \pi, \varphi \rrbracket = 1$.

The novelty of LTL[\mathcal{F}] is the ability to manipulate values by arbitrary functions. For example, \mathcal{F} may contain the quantitative operator ∇_λ , for $\lambda \in [0, 1]$, which tunes down the quality of a sub-specification. Formally, $\llbracket \pi, \nabla_\lambda \varphi_1 \rrbracket = \lambda \cdot \llbracket \pi, \varphi_1 \rrbracket$. Another useful operator is the weighted-average function \oplus_λ that is defined, for $\lambda \in [0, 1]$, by $\llbracket \pi, \varphi_1 \oplus_\lambda \varphi_2 \rrbracket = \lambda \cdot \llbracket \pi, \varphi_1 \rrbracket + (1 - \lambda) \cdot \llbracket \pi, \varphi_2 \rrbracket$. Consider, for example, the robot at the warehouse example from Section 1. Suppose shelf s_1 is at the east of the warehouse and we prefer the robot to be as close to the center as possible. Accordingly, we want a specification that incentivize the system to direct the robot in s_1 to the west, possibly also to the north or south, but less to the east. This can be done with the LTL[\mathcal{F}] specification $\psi_1 = \mathbf{G}(at_s_1 \rightarrow \mathbf{X}(W \vee \nabla_{\frac{4}{5}}(N \vee S) \vee \nabla_{\frac{3}{5}}E))$. Then, the satisfaction value of ψ_1 in computations in which the system directs the robot to go east from s_1 (for example, in order to satisfy other specifications), get satisfaction value $\frac{3}{5}$.

Suppose further that the robot sends a signal *low* whenever its battery falls below some threshold, in which case the system should direct the robot not to pick up new packages and to charge its battery in the first docking station it comes across. Ideally, the robot stays in this docking station for two consecutive time units. This can be stated with the LTL[\mathcal{F}] specification $\psi_2 = \mathbf{G}(low \rightarrow (\neg pickup \wedge \neg station)U(station \wedge (charging \oplus_{\frac{2}{3}} \mathbf{X}charging)))$. When the robot indeed stops at the first docking station and charges for two time units, the

satisfaction value is $\frac{2}{3} + \frac{1}{3} = 1$. If it stays there for only one time unit, the satisfaction value is $\frac{2}{3}$, and if it starts the charging only at the second time unit in the station, the satisfaction value drops to $\frac{2}{3}$. Note that the satisfaction value of ψ_1 and ψ_2 may not be 1 not only as a result of a non-optimal behavior but also as a result of hiding of an optimal behavior. For example, aiming to hide the secrets discussed in Section 1, the system may direct the robot to assign ? to *charging*, reducing the satisfaction value of ψ_2 .

► **Theorem 1** ([1]). *Let φ be an LTL[\mathcal{F}] formula over AP and $P \subseteq [0, 1]$ be a predicate. There exists an NGBW \mathcal{A}_φ^P over the alphabet 2^{AP} such that for every computation $\pi \in (2^{AP})^\omega$, we have that \mathcal{A}_φ^P accepts π iff $\llbracket \pi, \varphi \rrbracket \in P$. Furthermore, \mathcal{A}_φ^P has at most $2^{O(|\varphi|)}$ states and index at most $|\varphi|$.*

2.4 LTL[\mathcal{F}] synthesis

Consider finite disjoint sets I and O of input and output signals, which takes values in 2. For $i \in 2^I$ and $o \in 2^O$, let $i \cup o \in 2^{I \cup O}$ be the assignment that agrees with i and o . An I/O -transducer models an interaction between an environment that generates in each moment in time an input in 2^I and a system that responds with an output in 2^O . Formally, an I/O -transducer is a tuple $\mathcal{T} = \langle I, O, S, s_0, \eta, \tau \rangle$ where S is a finite set of states, $s_0 \in S$ is an initial state, $\eta : S \times 2^I \rightarrow S$ is a deterministic transition function, and $\tau : S \rightarrow 2^O$ is an output-labeling function. Given a sequence $w_I = i_0, i_1, i_2, \dots \in (2^I)^\omega$ of input letters, the *run of \mathcal{T} on w_I* is defined to be the sequence of states $r(w_I) = s_0, s_1, s_2, \dots \in S^\omega$ that begins with the initial state s_0 and is such that for all $j \geq 0$, we have $s_{j+1} = \eta(s_j, i_j)$. We define the *computation of \mathcal{T} on w_I* to be $\mathcal{T}(w_I) = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \dots \in (2^{I \cup O})^\omega$, where for all $j \geq 0$, we have $o_j = \tau(s_j)$. Note that the interaction is initiated by the system: the j -th output letter is determined by the j -th state, prior of reading the j -th input letter.

Defining the satisfaction value of φ in \mathcal{T} , denoted $\llbracket \mathcal{T}, \varphi \rrbracket$, the environment is assumed to be hostile and we care for the minimal satisfaction value of some computation of \mathcal{T} . Formally, $\llbracket \mathcal{T}, \varphi \rrbracket = \min\{\llbracket \mathcal{T}(w_I), \varphi \rrbracket : w_I \in (2^I)^\omega\}$. Note that no matter what the input sequence is, the specification φ is satisfied with value at least $\llbracket \mathcal{T}, \varphi \rrbracket$.

The *realizability* problem for LTL[\mathcal{F}] is to determine, given φ and a predicate $P \subseteq [0, 1]$, whether there exists a transducer \mathcal{T} such that $\llbracket \mathcal{T}, \varphi \rrbracket \in P$. We then say that \mathcal{T} realizes $\langle \varphi, P \rangle$. The *synthesis* problem is then to generate such a transducer. Of special interest are predicates P that are upward closed. Thus, $P = [v, 1]$ for some $v \in [0, 1]$.

2.5 Satisfaction value in noisy computations

Let the calligraphic digit 3 denote the set $\{\mathbf{F}, \mathbf{T}, ?\}$. We think of 3^{AP} as the set of *noisy assignments* to AP , where the truth value of a proposition mapped to ? is “unknown”. For two noisy assignments $\sigma, \sigma' \in 3^{AP}$, we say that σ' is *more informative* than σ , denoted $\sigma \leq_{\text{info}} \sigma'$, if for all $p \in AP$, we have that $\sigma(p) \in \{\sigma'(p), ?\}$. Thus, some assignments of \mathbf{F} and \mathbf{T} in σ' may become ? in σ . A *noisy computation* over AP is an infinite word $\kappa = \kappa_0, \kappa_1, \dots \in (3^{AP})^\omega$. We extend the \leq_{info} relation to noisy computations in the expected way, thus for $\kappa, \kappa' \in (3^{AP})^\omega$, we have that $\kappa \leq_{\text{info}} \kappa'$ iff for all $i \geq 0$, we have that $\kappa_i \leq_{\text{info}} \kappa'_i$.

A noisy assignment $\sigma \in 3^{AP}$ induces a set $\text{fill}(\sigma) \subseteq 2^{AP}$ of assignments, obtained by replacing assignments to ? by assignments to \mathbf{F} or \mathbf{T} . Formally, an assignment $\sigma' \in 2^{AP}$ is in $\text{fill}(\sigma)$ if $\sigma \leq_{\text{info}} \sigma'$. Each noisy computation κ induces a set $\text{fill}(\kappa)$ of computations in $(2^{AP})^\omega$, where $\pi = \pi_0, \pi_1, \dots$ is in $\text{fill}(\kappa)$ if for all $i \geq 0$, it holds that $\pi_i \in \text{fill}(\kappa_i)$. Note that $\kappa \leq_{\text{info}} \pi$ iff $\pi \in \text{fill}(\kappa)$.

For a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ and an LTL[\mathcal{F}] formula φ over AP , we denote by $\llbracket \kappa, \varphi \rrbracket$ the set of satisfaction values of φ in computations in $\text{fill}(\kappa)$. Formally,

$$\llbracket \kappa, \varphi \rrbracket = \{ \llbracket \pi, \varphi \rrbracket : \pi \in (\mathcal{Z}^{AP})^\omega \text{ is such that } \pi \in \text{fill}(\kappa) \}.$$

For an LTL formula ψ , we say that κ satisfies ψ , denoted $\kappa \models \psi$, if $\pi \models \psi$ for all computations π in $\text{fill}(\kappa)$. Thus, ψ is satisfied in all the computations obtained by filling κ . Note that for an LTL formula ψ , we have that $\llbracket \kappa, \psi \rrbracket$ is $\{0\}$, $\{1\}$, or $\{0, 1\}$. For simplicity, we use **T**, **F**, and **?** to refer to these cases. In particular, $\llbracket \kappa, \psi \rrbracket = ?$ if κ can be filled both to a computation that satisfies ψ and to a computation that does not satisfy ψ , and in such case we say that κ *hides* ψ .

3 Problem Formulation

In this section we define the problem of synthesis with privacy. We first define *noisy I/O-transducers*, which are the output of the synthesis algorithm.

3.1 Noisy transducers

A *noisy I/O-transducer* is $\mathcal{T} = \langle I, O, S, s_0, \eta, \tau, \mathbf{m} \rangle$, which augments an I/O-transducer by an *input-masking function* $\mathbf{m} : S \rightarrow 2^I$. In addition, the transition function assumes a noisy assignment to the input signals, thus $\eta : S \times \mathcal{Z}^I \rightarrow S$, and the labeling function generates a noisy assignment to the output signals, thus $\tau : S \rightarrow \mathcal{Z}^O$. Intuitively, when the transducer is in state s , it generates the noisy assignment $\tau(s)$ to the output signals and declares that the values of input signals in $\mathbf{m}(s)$ should stay private. Then, the environment generates an assignment $\sigma \in 2^I$ and reveals only the values of signals not in $\mathbf{m}(s)$. Thus, the transducer moves to the successor state $s' = \eta(s, \sigma')$, where $\sigma' \in \mathcal{Z}^I$ is obtained from σ by assigning **?** to the signals in $\mathbf{m}(s)$.

Formally, for an input assignment $i \in 2^I$ and a subset $M \in 2^I$ of I , let $\text{hide}(M, i) \in \mathcal{Z}^I$ be the noisy input assignment such that for every $p \in I$, if $p \in M$, then $\text{hide}(M, i)(p) = ?$, and if $p \notin M$, then $\text{hide}(M, i)(p) = i(p)$. Given an infinite sequence of assignments to the input signals $w_I = i_0, i_1, i_2, \dots \in (2^I)^\omega$, we define the *run* of \mathcal{T} on w_I and the *observable input sequence* induced by w_I , as the sequences $r(w_I) = s_0, s_1, s_2, \dots \in S^\omega$ and $w'_I = i'_0, i'_1, i'_2, \dots \in (\mathcal{Z}^I)^\omega$, respectively, where for all $j \geq 0$, we have that $i'_j = \text{hide}(\mathbf{m}(s_j), i_j)$ and $s_{j+1} = \eta(s_j, i'_j)$.

For a noisy input assignment $i' \in \mathcal{Z}^I$ and a noisy output assignment $o' \in \mathcal{Z}^O$, we define $i' \cup o' \in \mathcal{Z}^{I \cup O}$ as the noisy assignment that agrees with i' and o' . The *noisy computation* of \mathcal{T} on w_I is then $\mathcal{T}_{\mathbf{m}}(w_I) = (i'_0 \cup \tau(s_0)), (i'_1 \cup \tau(s_1)), (i'_2 \cup \tau(s_2)), \dots \in (\mathcal{Z}^{I \cup O})^\omega$.

Note that while each input sequence $w_I \in (2^I)^\omega$ induces a single noisy computation in $(\mathcal{Z}^{I \cup O})^\omega$, it induces several computations in $(2^{I \cup O})^\omega$. Namely, the set $\text{fill}(\mathcal{T}_{\mathbf{m}}(w_I))$ of all computations that are obtained by filling the noisy assignments to the signals that are unknown in $\mathcal{T}_{\mathbf{m}}(w_I)$.

3.2 Synthesis with privacy

In *synthesis with privacy*, we are given a specification φ in LTL[\mathcal{F}] and a set of secrets $\{\psi_1, \dots, \psi_k\}$ in LTL, and we seek a noisy I/O-transducer that satisfies φ in the highest specification value while keeping the satisfaction value of ψ_1, \dots, ψ_k unknown. Formally, a noisy I/O-transducer \mathcal{T} *realizes* $\langle \varphi, P \rangle$ with privacy ψ_1, \dots, ψ_k , for a predicate $P \subseteq [0, 1]$, if for every input sequence $w_I \in (2^I)^\omega$, it holds that $\llbracket \mathcal{T}_{\mathbf{m}}(w_I), \varphi \rrbracket \subseteq P$ and $\llbracket \mathcal{T}_{\mathbf{m}}(w_I), \psi_i \rrbracket = ?$, for all $1 \leq i \leq k$.

Note that we chose to focus on a setting where the secret ψ is an LTL (rather than $\text{LTL}[\mathcal{F}]$) formula. This is because the behaviors we want to keep private are typically qualitative. In Section 4.1 we describe how our framework can be extended to secrets in $\text{LTL}[\mathcal{F}]$.

Note also that while the input to our problem contains a single specification, it contains several secrets. Indeed, while for a set $\{\varphi_1, \dots, \varphi_k\}$ of specifications, a system realizes their conjunction $\varphi_1 \wedge \dots \wedge \varphi_k$ iff it realizes all conjuncts φ_i , for a set of secrets $\{\psi_1, \dots, \psi_k\}$, we cannot guarantee that the system hides all the secrets in the set by defining a single secret that is some Boolean combination of ψ_1, \dots, ψ_k . In particular, an unknown truth value for the conjunction $\psi_1 \wedge \dots \wedge \psi_k$ does not guarantee an unknown truth value for all conjuncts.

► **Remark 2.** Note that our framework hides the truth values of the secrets from an external observer: rather than observing computations in $(2^{I \cup O})^\omega$, it observes noisy computations in $(\mathcal{Z}^{I \cup O})^\omega$. If we want to assure the environment that the secrets are hidden also from the system, then we can change the framework so that the labeling function of the transducer generates non-noisy assignments to the output signals, thus $\tau : S \rightarrow 2^O$. Then, the result of the interaction is a computation in which only the input signals are noisy, and it should still keep the satisfaction values of the secrets unknown. Dually, if we only care about the privacy of the system, we can give up the noisy assignment to the input signals, which considerably simplifies the setting, as it makes the input-masking function unnecessary. ◀

4 Specifying Secrets

A key component of our algorithms is a construction of automata over an alphabet \mathcal{Z}^{AP} that accept noisy computations that hide the satisfaction value of a secret. In this section we define such automata. We start with secrets in LTL. Recall that a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ hides an LTL formula ψ if there are two computations $\pi_1, \pi_2 \in \text{fill}(\kappa)$ such that $\pi_1 \models \psi$ and $\pi_2 \not\models \psi$. Note that this implies that an observer of κ indeed does not know whether the computation that induces κ satisfies ψ . We first define an automaton that follows the above definition. Essentially, the automaton is obtained by taking the intersection of two automata, one that accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $1 \in \llbracket \kappa, \psi \rrbracket$, and one that accepts κ iff $0 \in \llbracket \kappa, \psi \rrbracket$.

► **Theorem 3.** *Let ψ be an LTL formula over AP. There exists an NGBW $\mathcal{N}_\psi^?$ over the alphabet \mathcal{Z}^{AP} such that for every noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$, we have that $\mathcal{N}_\psi^?$ accepts κ iff $\llbracket \kappa, \psi \rrbracket = ?$. Also, $\mathcal{N}_\psi^?$ has at most $2^{O(|\psi|)}$ states and index at most $|\psi|$.*

Proof. Let $\mathcal{A}_\psi^1 = \langle 2^{AP}, Q, Q_0, \delta, \alpha \rangle$ be an NGBW such that for every computation $\pi \in (2^{AP})^\omega$, it holds that \mathcal{A}_ψ^1 accepts π iff $\pi \models \psi$. Let $\mathcal{N}_\psi^T = \langle \mathcal{Z}^{AP}, Q, Q_0, \delta', \alpha \rangle$ be the NGBW obtained from \mathcal{A}_ψ^1 by letting it guess an assignment to atomic propositions whose value is unknown. Formally, for every state $q \in Q$ and letter $\sigma' \in \mathcal{Z}^{AP}$, we have that $\delta'(q, \sigma') = \bigcup \{ \delta(q, \sigma) : \sigma \in 2^{AP} \text{ is such that } \sigma' \leq_{\text{info}} \sigma \}$. It is easy to see to see that \mathcal{N}_ψ^T accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $1 \in \llbracket \kappa, \psi \rrbracket$. In a similar way, one can construct an NGBW \mathcal{N}_ψ^F that accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $0 \in \llbracket \kappa, \psi \rrbracket$. We can now define the required NGBW $\mathcal{N}_\psi^?$ as the intersection of \mathcal{N}_ψ^T and \mathcal{N}_ψ^F . By [42], both \mathcal{N}_ψ^T and \mathcal{N}_ψ^F have at most $2^{O(|\psi|)}$ states and index at most $|\psi|$, implying the same bound for $\mathcal{N}_\psi^?$. ◀

A drawback of the construction in Theorem 3 is that the constructed automaton is nondeterministic, which seems unavoidable. Indeed, it guesses values for the unknown signals that lead to satisfaction and violation of ψ . The use of a nondeterministic automaton makes it impossible to proceed with a Safraless synthesis algorithm, which requires universal

automata [31]. In order to address this weakness, we define a syntax-based three-valued semantics for LTL formulas when interpreted with respect to noisy computations. As we elaborate in the sequel, the syntax-based semantics coincides with the semantics-based one only for *well-specified* secrets, and it enables us to define universal automata for such secrets.

We start by defining the syntax-based three-valued semantics. We consider LTL formulas with the following syntax.

$$\psi := \mathbf{T} \mid p \mid \neg\psi_1 \mid \psi_1 \vee \psi_2 \mid \mathbf{X}\psi_1 \mid \psi_1 \mathbf{U}\psi_2.$$

Given a noisy computation $\kappa = \kappa_0, \kappa_1, \dots \in (\mathcal{Z}^{AP})^\omega$ and a position $j \geq 0$, we use κ^j to denote the suffix $\kappa_j, \kappa_{j+1}, \dots \in (\mathcal{Z}^{AP})^\omega$ of κ . The three-valued semantics maps a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ and an LTL formula ψ to the *three-valued satisfaction value of ψ in κ* , denoted $\langle\langle \kappa, \psi \rangle\rangle$, and defined inductively as follows.

- $\langle\langle \kappa, \mathbf{T} \rangle\rangle = \mathbf{T}$.
- $\langle\langle \kappa, p \rangle\rangle = \kappa_0(p)$.
- $\langle\langle \kappa, \neg\psi_1 \rangle\rangle = \neg\langle\langle \kappa, \psi_1 \rangle\rangle$, where $\neg\mathbf{T} = \mathbf{F}$, $\neg\mathbf{F} = \mathbf{T}$, and $\neg? = ?$.
- $\langle\langle \kappa, \psi_1 \vee \psi_2 \rangle\rangle = \begin{cases} \mathbf{T} & \text{if } \langle\langle \kappa, \psi_1 \rangle\rangle = \mathbf{T} \text{ or } \langle\langle \kappa, \psi_2 \rangle\rangle = \mathbf{T}, \\ \mathbf{F} & \text{if } \langle\langle \kappa, \psi_1 \rangle\rangle = \mathbf{F} \text{ and } \langle\langle \kappa, \psi_2 \rangle\rangle = \mathbf{F}, \\ ? & \text{otherwise.} \end{cases}$
- $\langle\langle \kappa, \mathbf{X}\psi_1 \rangle\rangle = \langle\langle \kappa^1, \psi_1 \rangle\rangle$.
- $\langle\langle \kappa, \psi_1 \mathbf{U}\psi_2 \rangle\rangle = \begin{cases} \mathbf{T} & \text{if } \exists i \geq 0. \langle\langle \kappa^i, \psi_2 \rangle\rangle = \mathbf{T} \text{ and } \forall 0 \leq j < i, \langle\langle \kappa^j, \psi_1 \rangle\rangle = \mathbf{T}, \\ \mathbf{F} & \text{if } \forall i \geq 0. \langle\langle \kappa^i, \psi_2 \rangle\rangle \neq \mathbf{F} \text{ implies } \exists 0 \leq j < i, \langle\langle \kappa^j, \psi_1 \rangle\rangle = \mathbf{F}. \\ ? & \text{otherwise.} \end{cases}$

As we now show, the classical translation of LTL formulas to NGBWs [42] can be extended to noisy computations. For an LTL formula ψ , let $cl(\psi)$ denote the set of ψ 's subformulas and their negation. The state space of our NGBW consists of functions $f \in \mathcal{Z}^{cl(\psi)}$ that do not contain propositional inconsistencies. For example, $f(\psi_1 \vee \psi_2) = ?$ iff $f(\psi_1) = ?$ and $f(\psi_2) \in \{?, \mathbf{F}\}$, or $f(\psi_2) = ?$ and $f(\psi_1) \in \{?, \mathbf{F}\}$. Then, the transition function corresponds to temporal requirements, and the acceptance condition makes sure that eventualities are not propagated forever. As is the case with the construction in [42], each noisy computation κ has a single accepting run in the NGBW: the run starts from the state f_0 that describes the satisfaction value of all the formulas in $cl(\psi)$ in κ (according to the syntax-based semantics), continues to the state f_1 that describes the satisfaction in the suffix κ^1 , and so on. Accordingly, the choice of initial states determines the language of the NGBW. For obtaining an NGBW for computations κ with $\langle\langle \kappa, \psi \rangle\rangle = ?$, we define the set of initial states to consists of functions f for which $f(\psi) = ?$. For obtaining an equivalent UGCW, we dualize the NGBW whose set of initial state consists of functions f for which $f(\psi) \neq ?$ (see proof in Appendix B.1).

► **Theorem 4.** *Let ψ be an LTL formula over AP. There exist an NGBW $\mathcal{S}_\psi^?$ and a UGCW $\mathcal{U}_\psi^?$ over the alphabet \mathcal{Z}^{AP} , such that for every noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$, we have that $\mathcal{S}_\psi^?$ accepts κ iff $\mathcal{U}_\psi^?$ accepts κ iff $\langle\langle \kappa, \psi \rangle\rangle = ?$. Also, $\mathcal{S}_\psi^?$ and $\mathcal{U}_\psi^?$ have at most $2^{O(|\psi|)}$ states and index at most $|\psi|$.*

The syntax-based semantics may not change the polarity of evaluations, yet it may lead to a loss of information. Formally, we have the following, which can be proved by an induction on the structure of the LTL formula.

► **Lemma 5.** *For every noisy computation κ and LTL formula ψ , if $\langle\langle \kappa, \psi \rangle\rangle \in \{\mathbf{F}, \mathbf{T}\}$, then $\langle\langle \kappa, \psi \rangle\rangle = \llbracket \kappa, \psi \rrbracket$. Possibly, however, $\langle\langle \kappa, \psi \rangle\rangle = ?$ and $\llbracket \kappa, \psi \rrbracket \in \{\mathbf{F}, \mathbf{T}\}$.*

We say that a secret ψ is *well-specified* if for all noisy computations κ , we have that $\llbracket \kappa, \psi \rrbracket = \langle \kappa, \psi \rangle$. Thus, the two semantics coincide for ψ . Equivalently, ψ is well-specified if $L(\mathcal{N}_\psi^?) = L(\mathcal{S}_\psi^?)$. The rationale behind the term “well-specified” is that, intuitively, the three-valued semantics loses information due to local dependencies that can be simplified. To see this, let us consider a few examples.

Recall that $\llbracket \kappa, \psi \rrbracket = \mathbf{T}$ if $\pi \models \psi$ for all computations $\pi \in \text{fill}(\kappa)$. Accordingly, for every noisy computation κ and tautology ψ and, we have that $\llbracket \kappa, \psi \rrbracket = \mathbf{T}$. In particular, $\llbracket \kappa, p \vee \neg p \rrbracket = \mathbf{T}$, even for a noisy computation κ with $\kappa_0(p) = ?$. On the other hand, for such a noisy computation κ , we have that $\langle \kappa, p \vee \neg p \rangle = ?$. This loss of information occurs not only with tautologies. For example, consider a noisy computation κ with $\kappa_0(q) = \mathbf{F}$ and $\kappa_0(p) = ?$. It is easy to see that $\llbracket \kappa, p \vee \neg(q \vee p) \rrbracket = \mathbf{T}$ whereas $\langle \kappa, p \vee \neg(q \vee p) \rangle = ?$. Moreover, the loss happens not only in the propositional level. Assume that κ above continues with $\kappa_1 = \kappa_0$ and consider the LTL formula $\psi = (p \wedge \mathbf{X}\neg p)\mathbf{U}q$. Note that $\llbracket \kappa, \psi \rrbracket = \mathbf{F}$ whereas $\langle \kappa, \psi \rangle = ?$.

Now, for the three examples above, we have that $p \vee \neg p = \mathbf{T}$, $p \vee \neg(q \vee p) = p \vee \neg q$, and $(p \wedge \mathbf{X}\neg p)\mathbf{U}q = q \vee (p \wedge \mathbf{X}(q \wedge \neg p))$, thus, all three formulas can be simplified to formulas that describe the intention of the designer in a clearer way. As is the case with other forms of *inherent vacuity* [26, 33], the fact that a secret is not well-specified is valuable information for the designer, as it points to redundant complications in the formulation of the secret. Theorems 3 and 4 are useful also for this task. To see this, consider an LTL formula ψ , and recall that, by definition, ψ is well-specified iff $L(\mathcal{N}_\psi^?) = L(\mathcal{S}_\psi^?)$. By Lemma 5, it is always the case that $L(\mathcal{N}_\psi^?) \subseteq L(\mathcal{S}_\psi^?)$. Thus, ψ is well-specified iff $L(\mathcal{S}_\psi^?) \subseteq L(\mathcal{N}_\psi^?)$. Note, however, that the above only gives us an EXPSPACE upper bound for the problem, and we leave the exact complexity open (see Section 7).

4.1 Extension to multiple and LTL[\mathcal{F}] secrets

The constructions above handle a single secret in LTL. In this section we show how to extend them to multiple and LTL[\mathcal{F}] secrets. We start with multiple secrets. Recall that a set $S = \{\psi_1, \dots, \psi_k\}$ of secrets cannot be composed to a single secret. Still, it is easy to extend the constructions above to such a set. First, in the semantics-based approach, we can extend Theorem 3 to S by taking an NGBW for the intersection language of the NGBWs $\mathcal{N}_{\psi_1}^?, \dots, \mathcal{N}_{\psi_k}^?$ described there, hence the following theorem.

► **Theorem 6.** *Let $S = \{\psi_1, \dots, \psi_k\}$ be a set of LTL formulas over AP. There exists an NGBW $\mathcal{N}_S^?$ over the alphabet 3^{AP} such that for every noisy computation $\kappa \in (3^{AP})^\omega$, we have that $\mathcal{N}_S^?$ accepts κ iff $\llbracket \kappa, \psi_i \rrbracket = ?$ for all $1 \leq i \leq k$. Also, $\mathcal{N}_S^?$ has at most $2^{O(m)}$ states and index at most m , where $m = \sum_{i=1}^k |\psi_i|$.*

Then, in the syntax-based approach, the situation is even simpler, as there we can actually compose S to a single secret. Indeed, under the syntax-based three valued semantics, for every noisy computation κ , we have that $\langle \kappa, \psi_i \vee \neg \psi_i \rangle = ?$ iff $\langle \kappa, \psi_i \rangle = ?$, and otherwise $\langle \kappa, \psi_i \vee \neg \psi_i \rangle = \mathbf{T}$. Accordingly, $\langle \kappa, (\psi_1 \vee \neg \psi_1) \vee \dots \vee (\psi_k \vee \neg \psi_k) \rangle = ?$ iff $\langle \kappa, \psi_i \rangle = ?$ for all $1 \leq i \leq k$. Thus, here, the fact $\psi_i \vee \neg \psi_i$ is a tautology and is thus not well-specified is surprisingly helpful.

We continue to LTL[\mathcal{F}] secrets. For two disjoint predicates $P_1, P_2 \subseteq [0, 1]$, we say that $\kappa \langle P_1, P_2 \rangle$ -hides ψ if there are two computations $\pi_1, \pi_2 \in \text{fill}(\kappa)$ such that $\llbracket \pi_1, \psi \rrbracket \in P_1$ and $\llbracket \pi_2, \psi \rrbracket \in P_2$. Thus, by observing κ , one cannot tell whether the satisfaction value of a computation that induces κ is in P_1 or P_2 . Note that the semantics-based definition for LTL is a special case of the above definition, with $P_1 = \{0\}$ and $P_2 = \{1\}$. It is not hard to see that the same construction described in the proof of Theorem 3 can be applied to LTL[\mathcal{F}] formulas, with the automata $\mathcal{A}_\psi^{P_1}$ and $\mathcal{A}_\psi^{P_2}$ (see Theorem 1) replacing the automata for ψ and $\neg\psi$ there. Formally, we have the following.

► **Theorem 7.** Let φ be an LTL[\mathcal{F}] formula over AP, and let $P_1, P_2 \subseteq [0, 1]$ be two predicates. There exists an NGBW $\mathcal{N}_\varphi^?$ over the alphabet 3^{AP} such that for every noisy computation $\kappa \in (3^{AP})^\omega$, we have that $\mathcal{N}_\varphi^?$ accepts κ iff $\kappa \langle P_1, P_2 \rangle$ -hides ψ . Also, $\mathcal{N}_\varphi^?$ has at most $2^{O(|\varphi|)}$ states and index at most $|\varphi|$.

Finally, handling a set S of LTL[\mathcal{F}] secrets combines Theorems 6 and 7: each secret ψ_i is given with predicates $P_1^i, P_2^i \subseteq [0, 1]$, and the NGBW $\mathcal{N}_S^?$ is obtained by intersecting these defined in Theorem 7.

5 Solving Synthesis with Privacy

In this section we describe a solution for the problem of synthesis with privacy. Let φ be an LTL[\mathcal{F}] formula (the specification), $P \subseteq [0, 1]$ a predicate, and ψ an LTL formula (the secret). Note that, for simplicity, we assume a single LTL secret. As described in Section 4.1, the extension to multiple and LTL[\mathcal{F}] secrets is easy. Consider a noisy computation $\kappa \in (3^{I \cup O})^\omega$. We say that κ is $\langle \psi, \varphi, P \rangle$ -good if $\llbracket \kappa, \varphi \rrbracket \subseteq P$ and $\llbracket \kappa, \psi \rrbracket = ?$. Recall that we seek a noisy transducer $\mathcal{T} = \langle I, O, S, s_0, \eta, \tau, \mathbf{m} \rangle$ such that for every input sequence $w_I \in (2^I)^\omega$, the noisy computations $\mathcal{T}_\mathbf{m}(w_I)$ is $\langle \psi, \varphi, P \rangle$ -good.

The next Theorem states that is possible to construct a DPW (and, in the case of well-specified secrets, also a UGCW) that recognizes $\langle \psi, \varphi, P \rangle$ -good noisy computations (see Appendix B.2). Once such a DPW or UGCW is defined, the problem is similar to usual synthesis, except that the transducer we construct is noisy and has to generate both noisy assignments to the output signals and input-masking instructions for the input signals.

► **Theorem 8.** Let φ be an LTL[\mathcal{F}] formula over AP, $P \subseteq [0, 1]$ a predicate, and ψ an LTL formula.

1. There exists a DPW $\mathcal{D}_{\varphi, \psi}^P$ over the alphabet 3^{AP} that recognizes $\langle \psi, \varphi, P \rangle$ -good noisy computations. The DPW $\mathcal{D}_{\varphi, \psi}^P$ has $2^{2^{O(|\varphi|+|\psi|)}}$ states and index $2^{O(|\varphi|+|\psi|)}$.
2. If ψ is well-specified, then there exists a UGCW $\mathcal{U}_{\varphi, \psi}^P$ over the alphabet 3^{AP} that recognizes $\langle \psi, \varphi, P \rangle$ -good noisy computations. The UGCW $\mathcal{U}_{\varphi, \psi}^P$ has $2^{O(|\varphi|+|\psi|)}$ states and index at most $|\varphi| + |\psi|$.

We proceed to define the notion of *noisy synthesis*, which refers to languages of noisy computations.

► **Definition 9.** Consider a language $L \subseteq (3^{I \cup O})^\omega$. We say that a noisy I/O-transducer \mathcal{T} realizes L if for all $w_I \in (2^I)^\omega$, the noisy computation $\mathcal{T}_\mathbf{m}(w_I)$ is in L . The noisy synthesis problem gets as input an automaton \mathcal{A} over the alphabet $3^{I \cup O}$ and returns a noisy I/O-transducer \mathcal{T} that realizes $L(\mathcal{A})$, or decides that no such transducer exists.

The next theorem follows immediately from the definition. Together with the constructions in Theorem 8, it enables us to reduce synthesis with privacy to noisy synthesis.

► **Theorem 10.** Consider an LTL[\mathcal{F}] specification φ , a predicate $P \subseteq [0, 1]$, and an LTL secret ψ . A noisy I/O-transducer \mathcal{T} realizes $\langle \varphi, P \rangle$ with privacy ψ iff \mathcal{T} realizes $L(\mathcal{D}_{\varphi, \psi}^P)$. When ψ is well-specified, then \mathcal{T} realizes $\langle \varphi, P \rangle$ with privacy ψ iff \mathcal{T} realizes $L(\mathcal{U}_{\varphi, \psi}^P)$.

Following Theorem 10, it is left to solve noisy synthesis for specifications given by a DPW or a UGCW. The algorithms are variants of these for traditional synthesis: For DPWs, we describe a reduction to deciding a parity game. For UGCWs, we describe a Safraless solution that is based on tree automata. In both solutions, we have to extend the solutions with

mechanisms that let the system choose the masked signals and direct the game or the tree automaton accordingly. Due to the lack of space, the definitions of tree automata can be found in Appendix A.

5.1 Solution for a DPW

In this section we describe a solution for the noisy-synthesis problem of a DPW $\mathcal{D} = \langle \mathcal{Z}^{I \cup O}, Q, q_0, \delta, \alpha \rangle$.

We reduce noisy synthesis of \mathcal{D} to the problem of finding a winning strategy in a parity game $\mathcal{G}_{\mathcal{D}}$ that models the interaction between the system (player SYS) and the environment (player ENV). At each round, SYS gives ENV masking instructions and a noisy output letter, and then ENV responds with a noisy input assignment according to the masking instructions of SYS. Formally, $\mathcal{G}_{\mathcal{D}} = \langle V, E, v_0, \alpha' \rangle$, where V is the set of positions and is partitioned into two disjoint sets $V = V_{\text{ENV}} \cup V_{\text{SYS}}$. The positions in $V_{\text{SYS}} = Q$ are controlled by SYS, and the positions in $V_{\text{ENV}} = Q \times 2^I \times \mathcal{Z}^O$ are controlled by ENV. The game starts in position $v_0 = q_0 \in V_{\text{SYS}}$, and it alternates between positions of SYS and ENV, i.e., $E \subseteq (V_{\text{SYS}} \times V_{\text{ENV}}) \cup (V_{\text{ENV}} \times V_{\text{SYS}})$. The exact definition of E is given by the following description of the possible moves in the game. For every $k \geq 0$ the k -th round of the game begins in a position $q_k \in V_{\text{SYS}}$ and proceeds as follows:

1. SYS chooses a noisy output assignment $o_k \in \mathcal{Z}^O$, and a set of input signals $M_k \in 2^I$, and the game moves to the position $\langle q_k, M_k, o_k \rangle \in V_{\text{ENV}}$.
2. ENV chooses an input assignment $i_k \in 2^I$, which is masked into $i'_k = \text{hide}(M_k, i_k)$, and the game moves to the position $q_{k+1} = \delta(q_k, i'_k \cup o_k) \in V_{\text{SYS}}$.

An outcome of the game then consists of the following components:

- a noisy input word $w'_I = i'_0, i'_1, i'_2, \dots \in (\mathcal{Z}^I)^\omega$,
- a noisy output word $w_O = o_0, o_1, o_2, \dots \in (\mathcal{Z}^O)^\omega$,
- a run $r = q_0, q_1, q_2, \dots \in Q^\omega$ of \mathcal{D} on $w'_I \cup w_O$.

Finally, the winning condition α' is induced by the acceptance condition α of \mathcal{D} ; thus a vertex v with Q -component q has $\alpha'(v) = \alpha(q)$.

We can now state the correctness of the reduction (see proof in Appendix B.3).

► **Proposition 11.** *The DPW \mathcal{D} is realizable by a noisy I/O-transducer iff SYS wins $\mathcal{G}_{\mathcal{D}}$.*

By Proposition 11, noisy synthesis of a DPW \mathcal{D} can be solved in the same complexity as the problem of deciding a parity game played on \mathcal{D} . Hence, by Theorem 8, we have the following (see proof in Appendix B.4). The lower bound follows from the fact we can reduce synthesis with privacy requirements to synthesis with no such requirements by adding a dummy atomic proposition $p \in I \cup O$ and a secret that refers to p .

► **Theorem 12.** *The problem of LTL[F] synthesis with privacy is 2EXPTIME-complete.*

5.2 Solution for a UGCW

In this section we describe a Safrless solution for the noisy-synthesis problem of a UGCW $\mathcal{U} = \langle \mathcal{Z}^{I \cup O}, Q, q_0, \delta, \alpha \rangle$.

We translate \mathcal{U} into a UGCT \mathcal{U}' on $2^I \times \mathcal{Z}^O$ -labeled \mathcal{Z}^I -trees that accept trees induced by noisy I/O-transducers that realize \mathcal{U} . We define $\mathcal{U}' = \langle \mathcal{Z}^I, \Sigma, Q, Q_0, \delta', \alpha \rangle$, where $\Sigma = 2^I \times \mathcal{Z}^O$, and $\delta' : Q \times \Sigma \rightarrow \mathcal{B}^+(\mathcal{Z}^I \times Q)$ is such that for every state $q \in Q$ and letter $\langle M, o \rangle \in \Sigma$, we have

$$\delta'(q, \langle M, o \rangle) = \bigwedge_{i \in 2^I} \bigwedge_{q' \in \delta(q, \text{hide}(M, i) \cup o)} \langle \text{hide}(M, i), q' \rangle$$

Note that if \mathcal{U}' is at node v labeled $\langle M, o \rangle$, and $i' \in 3^I$ is a noisy assignment such that $i'^{-1}(\{\mathbf{T}, \mathbf{F}\}) \neq M$, then \mathcal{U}' sends no requirements to the subtree that is the i' -successor of v . On the other hand, for a noisy assignment $i' \in (3^I)$ with $i'^{-1}(\{\mathbf{T}, \mathbf{F}\}) = M$, there is at least one copy that is sent to the i' -successor of v . This corresponds to the behavior of a noisy transducer: from a state s with $\mathbf{m}(s) = M$, the transducer is expected to handle every possible assignment to M , and when constructing a run, the assignments to signals not in $\mathbf{m}(s)$ are ignored.

Formally, we have the following (see proof in Appendix B.5).

► **Proposition 13.** *The UGCWU is realizable by a noisy I/O-transducer iff $L(\mathcal{U}') = \emptyset$.*

By Proposition 13, noisy synthesis of a UGCW \mathcal{U} can be reduced to the nonemptiness of a UGCT with the same state space and index. Hence, by [31] and Theorem 8, we have the following.

► **Theorem 14.** *The problem of LTL[\mathcal{F}] synthesis can be solved Safralessly in 2EXPTIME for well-specified secrets.*

6 On the Trade-off Between Utility and Privacy

Privacy involves loss of information, which makes it more difficult to realize specifications. Technically, missing information is quantified universally, and the realizing transducer has to satisfy the specification for all possible ways to fill it. In this section we discuss ways to examine and play with the trade off between utility, namely the satisfaction value of the specification φ , and privacy, namely the extent to which the satisfaction value of the secret ψ is revealed.

For secrets in LTL, which are Boolean, possible compensations on privacy include weakening of the secrets. One way to do it is to replace a secret ψ by a pair $\langle \theta, \psi \rangle$, indicating we care to keep the satisfaction value of ψ unknown only in noisy computations that satisfy θ . Note that, unlike the case of assumptions on the environment in synthesis [15], this cannot be achieved by changing the secret to $\theta \rightarrow \psi$. Indeed, the latter only means that we require the satisfaction value of $\theta \rightarrow \psi$ to be unknown. Our algorithms can be changed to address a $\langle \theta, \psi \rangle$ secret by replacing the automata constructed in Section 4 by ones that take θ into account, thus accept a noisy computation κ iff $\llbracket \kappa, \theta \rrbracket \neq \mathbf{T}$ or $\llbracket \kappa, \psi \rrbracket = ?$.

For secrets in LTL[\mathcal{F}], taking the predicates described in Section 4 to be closed upward, we can say, given $h \in (0, 1]$, that a noisy computation κ h -hides a secret ψ in LTL[\mathcal{F}] if $\max \llbracket \kappa, \psi \rrbracket - \min \llbracket \kappa, \psi \rrbracket \geq h$. Thus, knowing κ , our uncertainty about the satisfaction value of ψ is at least h . Note that LTL secrets are a special case of the above definition, with $h = 1$. Now, in synthesis with LTL[\mathcal{F}] secrets, the input includes, in addition to the specification ψ , a threshold v for it, and a secret ψ , also a threshold h for the secret, and we require the generated computation to both satisfy φ with value at least v and to h -hide ψ . Our algorithm can be changed to address the variants of the problem in which either v or h are given, and the goal is to maximize the other parameter, having the first one as a hard constraint. In particular, when h is given, we must h -hide φ , and seek a transducer that, under this constraint, maximizes the satisfaction value of φ . Technically, this amounts to replacing the DPW $\mathcal{D}_\psi^?$ by an automaton that accepts noisy computations that h -hides ψ . For the other case, where we fix v , the solution involves a search for h , which involves polynomially many executions of our algorithm.

7 Discussion

We introduced a simple yet powerful framework for synthesis of systems that preserve privacy. In our framework, the system and the environment may hide the values of signals they control, and they are guaranteed that “secrets” they care about are not going to be revealed. When one thinks about privacy, the first thing that comes to mind is privacy of *data* (age, salary, illnesses, gender, etc.). An underlying assumption of our work is that, in the context of reactive systems, privacy should concern *behaviours*. Thus, “secrets” are ω -regular languages, possibly weighted ones. A nice analogy is the way games are studied in the formal-methods community: classical game theory studies games with quantitative objectives, based on costs and rewards, whereas classical games in formal methods have ω -regular objectives, possibly weighted ones [9].

We introduced the key ideas behind the approach of “behavioral secrets”, namely a use of a three-valued semantics for the specification formalism. We also described how existing algorithms for synthesis, in fact even high-quality synthesis, can be extended to handle privacy. The latter is simple for traditional synthesis algorithms and involved a study of a syntax-based three-valued semantics for Safraless algorithms.

Beyond the challenge of extending the framework to richer settings of the synthesis problem (e.g., rational, distributed, infinite-state, or probabilistic systems [4, 25, 32, 38]), we find the following research directions, which address the basic idea of behavioral secrets, very interesting.

A stochastic approach. Recall that in the multi-valued setting, we followed the worst-case approach, thus the quality of the synthesized system is the minimal satisfaction value of the specification φ in some interaction. In the stochastic approach, we assume a given distribution on the input sequences, and the quality of the system is the expected satisfaction value of φ [2]. Extending synthesis with privacy to a stochastic approach, we seek noisy *I/O*-transducer that maximizes the expected satisfaction value of φ while hiding ψ with probability 1. Technically, as the valuation of φ refers to its expected satisfaction value, whereas hiding of the value of ψ is a hard constraint, the synthesis algorithm has to combine both types of objectives [3, 11, 6, 7]. The stochastic approach is of special interest when studying the trade-off between the expected uncertainty of ψ against the expected satisfaction of φ .

Specifying secrets. In our framework, a behavior ψ in LTL is kept secret if its satisfaction is unknown. More sophisticated definitions can refer to the *probability* that ψ is satisfied, given the revealed information, or, even more sophisticated, to the extent in which the revealed information changes the probability of ψ to be satisfied. For example, if the secret is $\psi = p \wedge q$ and we revealed that q holds, we still do not know the satisfaction value of ψ , yet we did learn that the probability of its satisfaction has increased. A good treatment of definitions that take probability in mind should address the fact that computations are sampled from the set of computations that satisfy the specification, which poses interesting technical challenges.

Multiple view-points. In our framework, revealed information is known to all parties: the system, the environment, and an observer to the interaction. In some settings, the environment is composed of several components who are willing to share information with the system, but not with each other. Also, not all components care about the satisfaction of all specifications. Such settings can be addressed by extending the framework to handle

multiple-viewpoint assignments to input and output signals. Thus, if the setting involves a set C of components, values are in $\{\mathbf{T}, \mathbf{F}\} \times 2^C$, specifying both the value and the subset of components that see it. Technically, the extension can be handled by using lattice automata and synthesis algorithms for them [28, 29].

Perturbation of signals. Our framework handles Boolean signals and allows the system and environment to hide the values of signals they control. In some settings, the Boolean signals encode richer values, or the setting includes non-Boolean inputs in the first place (e.g., augmenting LTL with Presburger arithmetic [19] or register automata with linear arithmetic over the rationals [16]). In such settings, it makes sense to allow the components not to entirely hide the value of their variables, but rather to *perturb* it to an approximated value. A synthesizing transducer should then perturb the value of the (non-Boolean) secret while satisfying the specification, possibly up to some perturbation.

Syntax-based three-valued semantic. As discussed in Section 4, our syntax-based three-valued semantic for LTL does not coincide with the semantics-based one. We described an EXPSPACE algorithm for deciding whether a given LTL formula is well-specified (that is, the two semantics coincide for it), and left the tight complexity of the problem open. Interestingly, the problem has similarities with both the satisfiability problem of \forall LTL, namely LTL augmented with universally-quantified propositions, which is EXPTIME-complete [40], and with *inherent vacuity*, namely deciding whether a given LTL formula ψ has a subformula θ such that ψ and $\forall x.\psi[\theta \leftarrow x]$ are equivalent, which is PSPACE-complete [26].

References

- 1 S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. *Journal of the ACM*, 63(3):24:1–24:56, 2016.
- 2 S. Almagor and O. Kupferman. High-quality synthesis against stochastic environments. In *Proc. 25th Annual Conf. of the European Association for Computer Science Logic*, volume 62 of *LIPICs*, pages 28:1–28:17, 2016.
- 3 S. Almagor, O. Kupferman, and Y. Verner. Minimizing expected cost under hard boolean constraints, with applications to quantitative synthesis. In *Proc. 27th Int. Conf. on Concurrency Theory*, volume 59 of *LIPICs*, pages 9:1–9:15, 2016.
- 4 C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, 3rd International Conference on Theoretical Computer Science*, volume 155 of *IFIP*, pages 493–506. Kluwer/Springer, 2004.
- 5 B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S.P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012.
- 6 R. Berthon, S. Guha, and J-F Raskin. Mixing probabilistic and non-probabilistic objectives in markov decision processes. In *Proc. 35th IEEE Symp. on Logic in Computer Science*, pages 195–208. ACM, 2020.
- 7 R. Berthon, M. Randour, and J-F. Raskin. Threshold constraints with guarantees for parity objectives in markov decision processes. In *Proc. 44th Int. Colloq. on Automata, Languages, and Programming*, volume 80 of *LIPICs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 8 R. Bloem, K. Chatterjee, T. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.
- 9 R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.

- 10 G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc. 11th Int. Conf. on Computer Aided Verification*, pages 274–287, 1999.
- 11 V. Bruyère, E. Filiot, M. Randour, and J-F. Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In *Proc. 31th Symp. on Theoretical Aspects of Computer Science*, volume 25 of *LIPICs*, pages 199–213, 2014.
- 12 C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th ACM Symp. on Theory of Computing*, pages 252–263, 2017.
- 13 P. Cerný, K. Chatterjee, T.A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *Proc. 23rd Int. Conf. on Computer Aided Verification*, pages 243–259, 2011.
- 14 K. Chatterjee, L. Doyen, T. A. Henzinger, and J-F. Raskin. Algorithms for ω -regular games with imperfect information. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 287–302, 2006.
- 15 K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th Int. Conf. on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008.
- 16 Y-F. Chen, O. Lengál, T. Tan, and Z. Wu. Register automata with linear arithmetic. In *Proc. 32nd IEEE Symp. on Logic in Computer Science*, pages 1–12, 2017.
- 17 M.R. Clarkson, B. Finkbeiner, M. Koleini, K.K. Micinski, M.N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *3rd International Conference on Principles of Security and Trust*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014.
- 18 L. de Alfaro and P. Roy. Solving games via three-valued abstraction refinement. *Inf. Comput.*, 208(6):666–676, 2010.
- 19 S. Demri. Linear-time temporal logics with presburger constraints: an overview. *Journal of Applied Non-Classical Logics*, 16(3-4):311–348, 2006.
- 20 I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the 22nd ACM Symposium on Principles of Database Systems*, pages 202–210. ACM, 2003.
- 21 J. Dubreil, Ph. Darondeau, and H. Marchand. Supervisory control for opacity. *IEEE Transactions on Automatic Control*, 55(5):1089–1100, 2010.
- 22 C. Dwork, F. McSherry, K. Nissim, and A.D. Smith. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 7(3):17–51, 2016.
- 23 E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ -calculus. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 267–278, 1986.
- 24 B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, and L. Tentrup. Synthesis from hyperproperties. *Acta Informatica*, 57(1-2):137–163, 2020.
- 25 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.
- 26 D. Fisman, O. Kupferman, S. Seinfeld, and M.Y. Vardi. A framework for inherent vacuity. In *4th International Haifa Verification Conference*, volume 5394 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2008.
- 27 S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016.
- 28 A. Gurfinkel and M. Chechik. Multi-valued model-checking via classical model-checking. In *Proc. 14th Int. Conf. on Concurrency Theory*, pages 263–277. Springer-Verlag, 2003.
- 29 O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2007.

- 30 O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
- 31 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.
- 32 M. Z. Kwiatkowska. Model checking and strategy synthesis for stochastic games: From theory to practice. In *Proc. 43th Int. Colloq. on Automata, Languages, and Programming*, volume 55 of *LIPICs*, pages 4:1–4:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 33 S. Maoz and R. Shalom. Inherent vacuity for GR(1) specifications. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 99–110. ACM, 2020.
- 34 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.
- 35 J.H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Science*, 29:274–301, 1984.
- 36 S. Safra. On the complexity of ω -automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.
- 37 S. Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Institute of Science, 1989.
- 38 S. Schewe. *Synthesis of distributed systems*. PhD thesis, Saarland University, Saarbrücken, Germany, 2008.
- 39 S. Shoham and O. Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In *Proc. 15th Int. Conf. on Computer Aided Verification*, volume 2725, pages 275–287, 2003.
- 40 A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
- 41 M.Y. Vardi. From verification to synthesis. In *VSTTE*, page 2, 2008.
- 42 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
- 43 Y. Wu, V. Raman, B.C. Rawlings, S. Lafortune, and S.A. Seshia. Synthesis of obfuscation policies to ensure privacy and utility. *Journal of Automated Reasoning*, 60(1):107–131, 2018.

A Tree Automata

Given a set D of directions, a D -tree is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T . For every $x \in T$, the nodes $x \cdot c$, for $c \in D$, are the *successors* of x . A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$. Given an alphabet Σ , a Σ -labeled D -tree is a pair $\langle T, \tau \rangle$ where T is a tree and $\tau : T \rightarrow \Sigma$ maps each node of T to a letter in Σ .

For a set X , let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using \wedge and \vee), where we also allow the formulas **T** and **F**. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **T** to elements in Y and assigning **F** to elements in $X \setminus Y$ makes θ true. An *alternating tree automaton* is $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α is an acceptance condition. We consider here the Büchi, co-Büchi, and parity acceptance conditions. For a state $q \in Q$, we use \mathcal{A}^q to denote the automaton obtained from \mathcal{A} by setting the initial state to be q . The *size* of \mathcal{A} , denoted $|\mathcal{A}|$, is the sum of lengths of formulas that appear in δ .

The alternating automaton \mathcal{A} runs on Σ -labeled D -trees. A *run* of \mathcal{A} over a Σ -labeled D -tree $\langle T, \tau \rangle$ is a $(T \times Q)$ -labeled \mathbb{N} -tree $\langle T_r, r \rangle$. Each node of T_r corresponds to a node of T . A node in T_r , labeled by $\langle x, q \rangle$, describes a copy of the automaton that reads the node x of T and visits the state q . Note that many nodes of T_r can correspond to the same node x of T . The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$.
2. Let $y \in T_r$ with $r(y) = \langle x, q \rangle$ and $\delta(q, \tau(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \dots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$, such that S satisfies θ , and for all $0 \leq i \leq n-1$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$.

For example, if $\langle T, \tau \rangle$ is a $\{0, 1\}$ -tree with $\tau(\varepsilon) = a$ and $\delta(q_{in}, a) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then, at level 1, the run $\langle T_r, r \rangle$ includes a node labeled $(0, q_1)$ or a node labeled $(0, q_2)$, and includes a node labeled $(0, q_3)$ or a node labeled $(1, q_2)$. Note that if, for some y , the transition function δ has the value **T**, then y need not have successors. Also, δ can never have the value **F** in a run.

A run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy the acceptance condition. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. That is, $inf(\pi)$ contains exactly all the states that appear infinitely often in π . The acceptance condition for alternating tree automata are similar to these defined for word automata, except that here, $inf(\pi)$ has to satisfy the condition α for all paths π . We denote by $L(\mathcal{A})$ the set of all Σ -labeled trees that \mathcal{A} accepts.

The alternating automaton \mathcal{A} is *nondeterministic* if for all the formulas that appear in δ , if (c_1, q_1) and (c_2, q_2) are conjunctively related, then $c_1 \neq c_2$. (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of $\{c\} \times Q$, for each $c \in D$, in each disjunct). The automaton \mathcal{A} is *universal* if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$, and \mathcal{A} is *deterministic* if it is both nondeterministic and universal. Note that word automata are a special case of tree automata, with $|D| = 1$.

B Missing Proofs

B.1 Proof of Theorem 4

For an LTL formula ψ , the *closure* of ψ , denoted $cl(\psi)$, is the set of ψ 's subformulas and their negation ($\neg\neg\psi$ is identified with ψ). Formally, $cl(\psi)$ is the smallest set of formulas that satisfy the following.

- $\psi \in cl(\psi)$.
- If $\psi_1 \in cl(\psi)$ then $\neg\psi_1 \in cl(\psi)$.
- If $\neg\psi_1 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$.
- If $\psi_1 \vee \psi_2 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$ and $\psi_2 \in cl(\psi)$.
- If $\mathbf{X}\psi_1 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$.
- If $\psi_1 \mathbf{U}\psi_2 \in cl(\psi)$ then $\psi_1 \in cl(\psi)$ and $\psi_2 \in cl(\psi)$.

Consider the set $cl(\psi)$. We say that a function $f \in \mathcal{F}^{cl(\psi)}$ is *consistent* if f does not have propositional inconsistency. Thus, f satisfies the following conditions.

1. For every formula $\psi_1 \in cl(\psi)$, one of the following holds:
 - $f(\psi_1) = \mathbf{T}$ and $f(\neg\psi_1) = \mathbf{F}$,
 - $f(\psi_1) = \mathbf{F}$ and $f(\neg\psi_1) = \mathbf{T}$, or
 - $f(\psi_1) = ?$ and $f(\neg\psi_1) = ?$.

2. For every formula of the form $\psi_1 \vee \psi_2 \in cl(\psi)$, the following holds.
- $f(\psi_1 \vee \psi_2) = \mathbf{T}$ iff $f(\psi_1) = \mathbf{T}$ or $f(\psi_2) = \mathbf{T}$.
 - $f(\psi_1 \vee \psi_2) = \mathbf{F}$ iff $f(\psi_1) = \mathbf{F}$ and $f(\psi_2) = \mathbf{F}$.
- Note that it follows that $f(\psi_1 \vee \psi_2) = ?$ iff $f(\psi_1) = ?$ and $f(\psi_2) \in \{?, \mathbf{F}\}$, or $f(\psi_2) = ?$ and $f(\psi_1) \in \{?, \mathbf{F}\}$.

Now, we define $\mathcal{S}_\psi^? = \langle \mathcal{Z}^{AP}, Q, \delta, Q_0, \alpha \rangle$, where

- The state space $Q \subseteq \mathcal{Z}^{cl(\psi)}$ is the set of all consistent functions.
 - Let f and f' be two states in Q , and let $\sigma \in \mathcal{Z}^{AP}$ be a letter. Then, $f' \in \delta(f, \sigma)$ if the following hold.
 1. For every $p \in AP$, we have that $\sigma(p) = f(p)$. Thus, σ agrees with f on the atomic propositions.
 2. For all $X\psi_1 \in cl(\psi)$, we have that $f(X\psi_1) = f'(\psi_1)$, and
 3. For all $\psi_1 U \psi_2 \in cl(\psi)$, we have
 - $f(\psi_1 U \psi_2) = \mathbf{T}$ iff $f(\psi_2) = \mathbf{T}$ or ($f(\psi_1) = \mathbf{T}$ and $f'(\psi_1 U \psi_2) = \mathbf{T}$).
 - $f(\psi_1 U \psi_2) = \mathbf{F}$ iff $f(\psi_2) = \mathbf{F}$ and ($f(\psi_1) = \mathbf{F}$ or $f'(\psi_1 U \psi_2) = \mathbf{F}$).
- Note that $f(\psi_1 U \psi_2) = ?$ iff one of the following hold:
- $f(\psi_2) = ?$ and ($f(\psi_1) \neq \mathbf{T}$ or $f'(\psi_1 U \psi_2) \neq \mathbf{T}$).
 - $f(\psi_2) = \mathbf{F}$, and $f(\psi_1) = \mathbf{T}$ and $f'(\psi_1 U \psi_2) = ?$.
 - $f(\psi_2) = \mathbf{F}$, and $f(\psi_1) = ?$ and $f'(\psi_1 U \psi_2) \neq \mathbf{F}$.
- $Q_0 \subseteq Q$ is the set of all states $f \in Q$ for which $f(\psi) = ?$.

- Every formula $\psi_1 U \psi_2$ contributes to α the two sets $\alpha_{\psi_1 U \psi_2}^{\mathbf{T}} = \{f \in Q : f(\psi_2) = \mathbf{T} \text{ or } f(\psi_1 U \psi_2) \neq \mathbf{T}\}$. and $\alpha_{\psi_1 U \psi_2}^? = \{f \in Q : f(\psi_2) = ? \text{ or } f(\psi_1 U \psi_2) \neq ?\}$.

Thus, if a run eventually visits only states in which the satisfaction value of $\psi_1 U \psi_2$ is \mathbf{T} , then it should visit infinitely many states in which the satisfaction value of ψ_2 is \mathbf{T} , and if a run eventually visits only states in which the satisfaction value of $\psi_1 U \psi_2$ is $?$, then it should visit infinitely many states in which the satisfaction value of ψ_2 is $?$.

Finally, $\mathcal{U}_\psi^?$ is obtained by dualizing the NGBW $\mathcal{S}_\psi^{?}$, which is similar to $\mathcal{S}_\psi^?$, except that $Q_0 \subseteq Q$ is the set of all states $f \in Q$ for which $f(\psi) \neq ?$.

B.2 Proof of Theorem 8

Given φ and P , let \bar{P} be the predicate that complements P , thus $\bar{P} = [0, 1] \setminus P$. By Theorem 1, we can construct an NGBW $\mathcal{A}_\varphi^{\bar{P}} = \langle \mathcal{Z}^{AP}, Q, Q_0, \delta, \alpha \rangle$ such that for every computation $\pi \in (\mathcal{Z}^{AP})^\omega$, it holds that $\mathcal{A}_\varphi^{\bar{P}}$ accepts π iff $\llbracket \pi, \varphi \rrbracket \notin P$. Also, $\mathcal{A}_\varphi^{\bar{P}}$ has at most $2^{O(|\varphi|)}$ states and index at most $|\varphi|$. Let $\mathcal{N}_\varphi^{\bar{P}} = \langle \mathcal{Z}^{AP}, Q, Q_0, \delta', \alpha \rangle$ be the NGBW obtained from $\mathcal{A}_\varphi^{\bar{P}}$ by letting it guess an assignment to atomic propositions whose value is unknown. Formally, for every state $q \in Q$ and letter $\sigma' \in \mathcal{Z}^{AP}$, we have that $\delta'(q, \sigma') = \bigcup \{ \delta(q, \sigma) : \sigma \in \mathcal{Z}^{AP} \text{ is such that } \sigma' \leq_{\text{info}} \sigma \}$. It is easy to see that $\mathcal{N}_\varphi^{\bar{P}}$ accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $\llbracket \kappa, \varphi \rrbracket \cap \bar{P} \neq \emptyset$. By dualizing $\mathcal{N}_\varphi^{\bar{P}}$, we get a UGCW \mathcal{U}_φ^P that accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $\llbracket \kappa, \varphi \rrbracket \subseteq P$.

By Theorem 3, given ψ , we can construct an NGBW $\mathcal{N}_\psi^?$ over the alphabet \mathcal{Z}^{AP} such that for every noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$, we have that $\mathcal{N}_\psi^?$ accepts κ iff $\llbracket \kappa, \psi \rrbracket = ?$. The NGBW $\mathcal{N}_\psi^?$ has at most $2^{O(|\psi|)}$ states and index at most $|\psi|$. Also, by Theorem 4, when ψ is well-specified, we can replace $\mathcal{N}_\psi^?$ by a UGCW $\mathcal{U}_\psi^?$.

Now, the desired UGCW $\mathcal{U}_{\varphi, \psi}^P$ can be obtained by taking the intersection of the UGCWs \mathcal{U}_φ^P and $\mathcal{U}_\psi^?$. Such an intersection does not involve a blow up (intersection of universal automata is dual to union of nondeterministic automata), and we end up with a UGCW with $2^{O(|\varphi| + |\psi|)}$ states and index at most $|\varphi| + |\psi|$.

In order to obtain the desired DPW $\mathcal{D}_{\varphi, \psi}^P$, we first co-determinize $\mathcal{N}_{\varphi}^{\bar{P}}$, and get a DPW \mathcal{D}_{φ}^P that accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $\llbracket \kappa, \varphi \rrbracket \subseteq P$. By [36, 34], the DPW \mathcal{D}_{φ}^P has $2^{2^{O(|\varphi|)}}$ states and index $2^{O(|\varphi|)}$. Then, we determinize $\mathcal{N}_{\psi}^?$ and get a DPW $\mathcal{D}_{\psi}^?$ with at most $2^{2^{O(|\psi|)}}$ states and index $2^{O(|\psi|)}$ such that $\mathcal{D}_{\psi}^?$ accepts a noisy computation $\kappa \in (\mathcal{Z}^{AP})^\omega$ iff $\llbracket \kappa, \psi \rrbracket = \{0, 1\}$. The DPW $\mathcal{D}_{\varphi, \psi}^P$ is then obtained by taking the intersection of \mathcal{D}_{φ}^P and $\mathcal{D}_{\psi}^?$. Since intersection of DPWs involve an exponential blow up only in their indices, the required bounds on the state space and index follows.

In more detail, Parity automata can be translated into Street automata on top of the same structure and with index of the same order. Thus, we may treat both automata as Street automata of size and index of the same order. Then, we take the intersection DSW which is of size $2^{k_\varphi + k_\psi}$ and index $k_\varphi + k_\psi$. By [37], a deterministic Street automaton with m states and index k can be translated into a deterministic Rabin automaton with $\Theta(m2^{k \log k})$ states and index $t = \Theta(k)$. The pairs in the acceptance condition in the Rabin automaton $((B_i, G_i))_{i=1}^t$ are such that $B_i \subseteq B_j$ for all $i \leq j$ and all of the G_i are disjoint. Thus, it is not hard to see that the parity condition that gives G_i priority $2i$, and $B_i \setminus B_{i-1}$ priority $2i - 1$, and all other states priority $2t + 1$, defines an equivalent deterministic parity automaton, with states and index of the same order as the Rabin automaton. Hence, the DPW \mathcal{A} for the intersection language has $2^{k_\varphi + k_\psi} 2^{(k_\varphi + k_\psi) \log(k_\varphi + k_\psi)} \leq 2^{2^{O(|\varphi| + |\psi|)}}$ states and index $O(k_\varphi + k_\psi) \leq 2^{O(|\varphi| + |\psi|)}$.

B.3 Proof of Proposition 11

We partition the proposition into two propositions.

► **Proposition 15.** *If $\mathcal{G}_{\mathcal{D}}$ is winning for SYS, then a noisy I/O-transducer \mathcal{T} that realizes \mathcal{D} can be constructed on top of \mathcal{D} in time $O(n^k)$, where n is the number of positions in $\mathcal{G}_{\mathcal{D}}$ and k is the index of \mathcal{D} .*

Proof. Since parity games enjoy memoryless-determinacy, it follows that SYS wins iff it has a memoryless strategy. Thus assume that SYS wins $\mathcal{G}_{\mathcal{D}}$ and let $f_{\text{SYS}} : V_{\text{SYS}} \rightarrow V_{\text{ENV}}$ be a winning memoryless strategy for SYS. Note that such a winning memoryless strategy f_{SYS} can be computed in time $O(n^k)$ [23] (in fact less, using improved algorithms for parity games [12]). We define a noisy I/O-transducer \mathcal{T} as follows. The set of states of the transducer \mathcal{T} is $S = V_{\text{SYS}} = Q$. For a state $q \in S$, let $f_{\text{SYS}}(q) = \langle q, M, o \rangle$, we set $\tau(q) = o$ and $\mathbf{m}(q) = M$. Then, for $i' \in \mathcal{Z}^I$ for which there exists $i \in \mathcal{Z}^I$ with $i' = \text{hide}(M, i)$, we define the transition function by $\eta(q, i') = \delta(q, i' \cup o)$, and otherwise, if there is no such $i \in \mathcal{Z}^I$, then we define $\eta(q, i')$ to be an arbitrary state (Recall that runs of \mathcal{T} does not use such transitions of η). In other words, we let ENV play with $i \in \mathcal{Z}^I$ from $\langle q, M, o \rangle$, and move to the appropriate i -successor in the game. Notice that for all $w_I \in (\mathcal{Z}^I)^\omega$ the computation $\mathcal{T}_{\mathbf{m}}(w_I) = (i'_0 \cup o_0), (i'_1 \cup o_1), \dots \in (\mathcal{Z}^{I \cup O})^\omega$ is obtained from the input and output components of the outcome of the game $\mathcal{G}_{\mathcal{D}}$ when ENV plays with $w_I = i_0, i_1, i_2, \dots \in (\mathcal{Z}^I)^\omega$ and SYS plays according to the strategy f_{SYS} . Hence, since f_{SYS} is winning for the System, it follows that for all $w_I \in (\mathcal{Z}^I)^\omega$, the run of \mathcal{D} over $\mathcal{T}_{\mathbf{m}}(w_I)$ is accepting. That is, \mathcal{T} is a noisy I/O-transducer that realizes \mathcal{D} . ◀

► **Proposition 16.** *If \mathcal{D} is realizable with a noisy I/O-transducer, then SYS wins $\mathcal{G}_{\mathcal{D}}$.*

Proof. Assume that $\mathcal{T} = \langle I, O, \mathcal{L}, S, \eta, \tau, \mathbf{m} \rangle$ is a noisy I/O-transducer that realizes \mathcal{D} , we will construct a winning strategy f_{SYS} that uses \mathcal{T} as a memory structure. Let W be the set of all finite paths in $\mathcal{G}_{\mathcal{D}}$ that start in $v_0 = q_0 \in V_{\text{SYS}}$ and end in some position $v_k \in V_{\text{SYS}}$ that belongs

to SYS. We define the strategy $f_{\text{SYS}} : W \rightarrow V_{\text{ENV}}$ as a partial function, where f_{SYS} is defined on $\langle q_0 \rangle \in W$, and for all $\rho = \langle q_0, \langle q_0, M_0, o_0 \rangle, q_1, \dots, \langle q_{k-1}, M_{k-1}, o_{k-1} \rangle, q_k \rangle \in W$, if f_{SYS} is defined on ρ , and $f_{\text{SYS}}(\rho) = \langle q_k, M_k, o_k \rangle$, then for all $i \in 2^I$, if $q_{k+1} = \delta(q_k, \text{hide}(M_k, i) \cup o_k)$, then f_{SYS} is also defined on $\rho' = \langle q_0, \langle q_0, M_0, o_0 \rangle, \dots, \langle q_k, M_k, o_k \rangle, q_{k+1} \rangle \in W$. Namely, f_{SYS} is defined on ρ' , which is the extension of ρ when *Sys* plays with f_{SYS} , hence moves to $f_{\text{SYS}}(\rho) = \langle q_k, M_k, o_k \rangle$, and then ENV proceeds to $q_{k+1} = \delta(q_k, \text{hide}(M_k, i) \cup o_k)$ for some $i \in 2^I$. In order to define f_{SYS} we also define two more partial functions $f_S : W \rightarrow S$ and $f_I : W \rightarrow 2^I$. Intuitively, f_I guesses the last input letter played by ENV, and f_S simulates the run of \mathcal{T} on the word guessed by f_I . The functions f_S and f_I have the same domain as f_{SYS} , with the only exception that f_I is not defined on the path $\rho = \langle q_0 \rangle$, as ENV haven't yet played, and hence there's nothing for f_I to guess. We define f_S , f_I and f_{SYS} by induction. First, for $\rho = \langle q_0 \rangle$, let $f_S(\rho) = s_0$, where $s_0 \in S$ is the initial state of \mathcal{T} , and let $f_{\text{SYS}}(\rho) = \langle q_0, \mathbf{m}(f_S(q_0)), \tau(f_S(q_0)) \rangle$. Then, assume that f_S and f_{SYS} have been defined on $\rho = \langle q_0, \langle q_0, M_0, o_0 \rangle, q_1, \dots, \langle q_{k-1}, M_{k-1}, o_{k-1} \rangle, q_k \rangle \in W$, and let $f_{\text{SYS}}(\rho) = \langle q_k, M_k, o_k \rangle \in V_{\text{ENV}}$. Consider $q_{k+1} \in V_{\text{SYS}}$ such that $(f_{\text{SYS}}(\rho), q_{k+1}) \in E$. I.e., q_{k+1} is a possible move of ENV from $f_{\text{SYS}}(\rho) = \langle q_k, M_k, o_k \rangle$. Let $i_k \in 2^I$ be some input letter such that $q_{k+1} = \delta(q_k, \text{hide}(M_k, i_k) \cup o_k)$. Note that such an input letter $i_k \in 2^I$ exists since q_{k+1} is a successor of the ENV-position $f_{\text{SYS}}(\rho) = \langle q_k, M_k, o_k \rangle$. Thus for the extension $\rho' = \langle q_0, \langle q_0, M_0, o_0 \rangle, \dots, \langle q_k, M_k, o_k \rangle, q_{k+1} \rangle \in W$ of ρ , we set $f_I(\rho') = i_k$, and $f_S(\rho') = \eta(f_S(\rho), \text{hide}(M_k, i_k))$ and $f_{\text{SYS}}(\rho') = \langle q_{k+1}, \mathbf{m}(f_S(\rho')), \tau(f_S(\rho')) \rangle$. It is now not hard to see that any outcome of the game when SYS plays with f_{SYS} , is such that the run component $r_{\mathcal{D}}$ is a run of \mathcal{D} over the noisy computation $\mathcal{T}_{\mathbf{m}}(w_I)$, where $w_I = i_0, i_1, i_2, \dots \in (2^I)$ is obtained by f_I . Hence, since \mathcal{T} realizes \mathcal{D} , it follows that $r_{\mathcal{D}}$ is accepting. That is, any outcome of the game when SYS plays with f_{SYS} is winning for SYS, and f_{SYS} is a winning strategy for SYS. \blacktriangleleft

B.4 Proof of Theorem 12

We start with the upper bound. Given an LTL[\mathcal{F}] specification φ , a predicate $P \subseteq [0, 1]$, and an LTL secret ψ , we construct the DPW $\mathcal{D} = \mathcal{D}_{\varphi, \psi}^P$ as in Theorem 8, and then solve the game $\mathcal{G}_{\mathcal{D}}$. By Theorem 10 and Proposition 11, it follows that $\langle \varphi, P \rangle$ is realizable with privacy ψ iff SYS wins $\mathcal{G}_{\mathcal{D}}$, and that solving $\mathcal{G}_{\mathcal{D}}$ is done in time $O(n^k)$ where n is the number of positions in $\mathcal{G}_{\mathcal{D}}$ and k is the index of \mathcal{D} . By Theorem 8, the number of states in \mathcal{D} is $|Q| = 2^{2^{O(|\varphi|+|\psi|)}}$, and the index is of size $k = 2^{O(|\varphi|+|\psi|)}$, and in particular, the construction of \mathcal{D} is done in 2EXPTIME in the size of the formulas φ and ψ . The number of positions in $\mathcal{G}_{\mathcal{D}}$ is $|V| \leq |Q| \cdot 3^{|I|+|O|} = 2^{2^{O(|\varphi|+|\psi|)}} \cdot 3^{|I|+|O|}$, and the number of priorities is the same as in \mathcal{D} . We may assume that $I \cup O \subseteq \text{cl}(\varphi) \cup \text{cl}(\psi)$, hence $3^{|I|+|O|} = 2^{O(|\varphi|+|\psi|)}$, and $|V| = 2^{2^{O(|\varphi|+|\psi|)}}$. Thus, $\mathcal{G}_{\mathcal{D}}$ is solved in time,

$$n^k \leq (2^{2^{O(|\varphi|+|\psi|)}})^{2^{O(|\varphi|+|\psi|)}} = 2^{2^{O(|\varphi|+|\psi|)}}$$

That is, $\mathcal{G}_{\mathcal{D}}$ is solved in 2EXPTIME in the size of φ and ψ .

For the lower bound, it is easy to reduce LTL[\mathcal{F}] synthesis with no privacy requirements to LTL[\mathcal{F}] synthesis with such requirements, for example by adding a secret that refers to a dummy output signal $p \notin I \cup O$.

B.5 Proof of Proposition 13

We prove that if $L(\mathcal{U}') = \emptyset$ then \mathcal{U}' is not realizable by a noisy I/O -transducer, and that if $L(\mathcal{U}') \neq \emptyset$, then there is a finite witness for the nonemptiness of \mathcal{U}' that encodes a noisy transducer that realizes \mathcal{U} .

Given a $(2^I \times 3^O)$ -labeled 3^I -tree $\langle (3^I)^*, f \rangle$ and an input word $w_I = i_0, i_1, i_2, \dots \in (2^I)^\omega$, we define the sequence of masking instructions $M_0, M_1, M_2, \dots \in (2^I)^\omega$, the sequence of noisy output assignments $o_0, o_1, o_2, \dots \in (3^O)^\omega$, and the masked input word $w'_I = i'_0, i'_1, i'_2, \dots \in (3^I)^\omega$ that correspond to f and w_I as follows. First, $\langle M_0, o_0 \rangle = f(\varepsilon)$. Then, for all $k \geq 0$, we have that $i'_k = \text{hide}(M_k, i_k)$ and $\langle M_{k+1}, o_{k+1} \rangle = f(i'_0, i'_1, \dots, i'_k)$. Then, let $\kappa = (i'_0 \cup o_0), (i'_1 \cup o_1), \dots \in (3)^{I \cup O}$ be the noisy computation that correspond to f and w_I . Observe that f is accepted by \mathcal{U}' iff for all $w_I \in (2^I)^\omega$, the noisy computation κ that corresponds to f and w_I is accepted by \mathcal{U} . Thus, f can be thought as a strategy for the noisy synthesis of \mathcal{U} , and f is accepted by \mathcal{U}' iff it is a winning strategy.

Note that the language of \mathcal{U}' is not empty iff there is a finite memory strategy $f : (3^I)^* \rightarrow 2^I \times 3^O$ that is accepted by \mathcal{A}' , and the memory structure of f is at most exponential in the size of \mathcal{A}' [31]. Hence, the specification given by \mathcal{A} is realizable by a noisy I/O -transducer iff the language of \mathcal{A}' is not empty, and a finite memory witness for the non-emptiness of \mathcal{A}' is a noisy I/O -transducer that realizes \mathcal{A} . Deciding whether the language of a UGCT is empty, and finding a finite memory witness in the case it is not empty is in EXPTIME. Hence, the synthesis of a noisy transducer that realizes \mathcal{A} is reduced to the nonemptiness of UGCT problem, and we have an EXPTIME upper bound.

A Generic Polynomial Time Approach to Separation by First-Order Logic Without Quantifier Alternation

Thomas Place  

LaBRI, Bordeaux University, France

Marc Zeitoun   

LaBRI, Bordeaux University, France

Abstract

We look at classes of languages associated to fragments of first-order logic $\mathcal{B}\Sigma_1$, in which quantifier alternations are disallowed. Each such fragment is fully determined by choosing the set of predicates on positions that may be used. Equipping first-order logic with the linear ordering and possibly the successor relation as predicates yields two natural fragments, which were investigated by Simon and Knast, who proved that these two variants have decidable *membership*: “does an input regular language belong to the class?”. We extend their results in two orthogonal directions.

- First, instead of membership, we explore the more general separation problem: decide if two regular languages can be separated by a language from the class under study.
- Second, we use more general inputs: classes \mathcal{G} of *group languages* (*i.e.*, recognized by a DFA in which each letter induces a permutation of the states) and extensions thereof, written \mathcal{G}^+ .

We rely on a characterization of $\mathcal{B}\Sigma_1$ by the operator $BPol$: given an input class \mathcal{C} , it outputs a class $BPol(\mathcal{C})$ that corresponds to a variant of $\mathcal{B}\Sigma_1$ equipped with special predicates associated to \mathcal{C} . The classes $BPol(\mathcal{G})$ and $BPol(\mathcal{G}^+)$ capture many natural variants of $\mathcal{B}\Sigma_1$ which use predicates such as the linear ordering, the successor, the modular predicates or the alphabetic modular predicates.

We show that separation is decidable for $BPol(\mathcal{G})$ and $BPol(\mathcal{G}^+)$ when this is the case for \mathcal{G} . This was already known for $BPol(\mathcal{G})$ and for two particular classes of the form $BPol(\mathcal{G}^+)$. Yet, the algorithms were indirect and relied on involved frameworks, yielding poor upper complexity bounds. In contrast, our approach is direct. We work only with elementary concepts (mainly, finite automata). Our main contribution consists in polynomial time Turing reductions from both $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation to \mathcal{G} -separation. This yields polynomial time algorithms for several key variants of $\mathcal{B}\Sigma_1$, including those equipped with the linear ordering and possibly the successor and/or the modular predicates.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Regular languages

Keywords and phrases Automata, Separation, Covering, Concatenation hierarchies, Group languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.43

Related Version *Full Version*: <https://arxiv.org/abs/2210.00946> [25]

Funding Supported by the DeLTA project (ANR-16-CE40-0007).

1 Introduction

An important question in automata theory is to precisely understand the prominent classes of regular languages of finite words. We are interested in the classes associated to a piece of syntax (such as regular expressions or logic), whose purpose is to specify the languages of such classes. In the paper, we formalize the goal of “understanding a given class \mathcal{C} ” by looking at a decision problem: \mathcal{C} -separation. It takes two regular languages L_1, L_2 as input and asks whether there exists $K \in \mathcal{C}$ such that $L_1 \subseteq K$ and $K \cap L_2 = \emptyset$. The key idea is that obtaining an algorithm for \mathcal{C} -separation requires a solid understanding of \mathcal{C} .



© Thomas Place and Marc Zeitoun;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 43; pp. 43:1–43:22



Leibniz International Proceedings in Informatics

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

We investigate a family of classes associated to a fragment of first-order logic written $\mathcal{BS}\Sigma_1$. The sentences of $\mathcal{BS}\Sigma_1$ are Boolean combinations of *existential* formulas, *i.e.*, whose prenex normal form has the shape $\exists x_1 \exists x_2 \cdots \exists x_k \varphi$, with φ quantifier-free. Several classes are associated to $\mathcal{BS}\Sigma_1$, each determined by the predicates on positions that we allow. In the literature, standard examples of predicates include the linear order “ $<$ ” [27], the successor relation “ $+1$ ” [9] or modular predicates “ MOD ” [5]. Thus, a generic approach is desirable.

We tackle languages associated to $\mathcal{BS}\Sigma_1$ through the operator $\mathcal{C} \mapsto BPol(\mathcal{C})$ defined on classes of languages. It is the composition of the polynomial closure $\mathcal{C} \mapsto Pol(\mathcal{C})$ and the Boolean closure $\mathcal{C} \mapsto Bool(\mathcal{C})$ operators: $BPol(\mathcal{C}) = Bool(Pol(\mathcal{C}))$. Recall that the polynomial closure of a class \mathcal{C} consists of all finite unions of languages of the form $L_0 a_1 L_1 \cdots a_n L_n$, where $n \geq 0$, each a_i is a letter and each L_i belongs to \mathcal{C} . Indeed, many classes associated to $\mathcal{BS}\Sigma_1$ are of the form $BPol(\mathcal{C})$ [34, 20]. In this paper, we look at specific input classes \mathcal{C} .

The *group languages* are those recognized by a finite group, or equivalently by a permutation automaton [33] (*i.e.*, which is complete, deterministic *and* co-deterministic). We consider input classes that are either a class \mathcal{G} consisting of group languages, or a well-suited extension thereof, \mathcal{G}^+ (roughly, \mathcal{G}^+ is the least Boolean algebra containing \mathcal{G} and the singleton $\{\varepsilon\}$). It is known [20] that if \mathcal{G} is a class of group languages, then $BPol(\mathcal{G}) = \mathcal{BS}\Sigma_1(<, \mathbb{P}_{\mathcal{G}})$ and $BPol(\mathcal{G}^+) = \mathcal{BS}\Sigma_1(<, +1, \mathbb{P}_{\mathcal{G}})$. Here, $\mathbb{P}_{\mathcal{G}}$ is a set of predicates associated to \mathcal{G} : each language L in \mathcal{G} gives rise to a predicate $P_L(x)$, which selects all positions x in a word w such that the prefix of w up to position x (excluded) belongs to L . This captures most of the natural examples. In particular, we get signatures including the aforementioned predicates, such as $\{<\}$, $\{<, +1\}$, $\{<, MOD\}$ and $\{<, +1, MOD\}$ (we provide some more examples in the paper).

State of the art. Historically, $BPol(\mathcal{G})$ and $BPol(\mathcal{G}^+)$ were first investigated for particular input classes. A prominent example is the class of piecewise testable languages [27], *i.e.*, the class $BPol(ST) = \mathcal{BS}\Sigma_1(<)$ where $ST = \{\emptyset, A^*\}$. It was shown that $BPol(ST)$ -separation is decidable in [1] using technical algebraic arguments. Simpler polynomial time algorithms were discovered later [17, 6]. There also exists an involved specialized separation algorithm [36] for $BPol(MOD) = \mathcal{BS}\Sigma_1(<, MOD)$, where MOD is the class of modulo languages. Decidability can be lifted to $BPol(ST^+) = \mathcal{BS}\Sigma_1(<, +1)$ (the languages of dot-depth one [9]) and to $BPol(MOD^+) = \mathcal{BS}\Sigma_1(<, +1, MOD)$ via transfer results [22, 16]. Unfortunately, this approach yields an exponential complexity blow-up. Recently, a generic approach was developed for $BPol(\mathcal{G})$. It is proved in [21] that if \mathcal{G} is a class of group languages with mild hypotheses, $BPol(\mathcal{G})$ -separation is decidable when \mathcal{G} -separation is decidable. Yet, this generic approach is indirect and considers a more general problem: *covering*. Because of this, the algorithms and their proofs are complex and rely on an intricate framework [19], yielding poor upper complexity bounds. This contrasts with the simple polynomial time procedures presented in [17, 6] for $BPol(ST)$. No generic result of this kind is known for the classes $BPol(\mathcal{G}^+)$.

Contributions. We give *generic polynomial time Turing reductions* from $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation to \mathcal{G} -separation, where \mathcal{G} is a class of group languages with mild properties. We present them as greatest fixpoint procedures which use an oracle for \mathcal{G} -separation at each step and run in *polynomial time* (for input languages represented by nondeterministic finite automata). While the proofs are involved, they are self-contained and based exclusively on elementary concepts from automata theory. No particular knowledge on group theory is required to follow them: we only use the very definition of a group.

For $BPol(\mathcal{G})$, this new approach is a significant improvement on the results of [21]. While we do reuse some ideas of [21], we complement them with new ones and the presentation is independent. We get a simpler algorithm, which requires only basic notions from automata

theory. In particular, one direction of the proof describes a generic construction for building separators in $BPol(\mathcal{G})$ (when they exist). This serves our main objective: understanding classes of languages. In addition, we obtain much better complexity upper bounds on $BPol(\mathcal{G})$ -separation. Finally, our techniques can handle $BPol(\mathcal{G}^+)$ as well. This was not the case in [21]: the generic reduction from $BPol(\mathcal{G}^+)$ -separation to \mathcal{G} -separation is a new result.

These results apply to several key classes. Separation is decidable in polynomial time for $ST = \{\emptyset, A^*\}$, for the class MOD of modulo languages and for the class GR of *all* group languages [26]. Hence, the problem is also decidable in polynomial time for $BPol(ST)$ (i.e., $\mathcal{B}\Sigma_1(<)$), $BPol(ST^+)$ (i.e., $\mathcal{B}\Sigma_1(<, +1)$), $BPol(MOD)$ (i.e., $\mathcal{B}\Sigma_1(<, MOD)$), $BPol(MOD^+)$ (i.e., $\mathcal{B}\Sigma_1(<, +1, MOD)$), $BPol(GR)$ and $BPol(GR^+)$ (the logical characterization of the last two classes is not standard, yet they are quite prominent as well [11, 8]). This reproves a known result for $BPol(ST)$ (in fact, we essentially reprove the algorithm of [6]). The polynomial time upper bounds are new for all other classes. Another application is the class AMT of alphabet modulo testable languages (which are recognized by commutative groups): $BPol(AMT)$ and $BPol(AMT^+)$ correspond to $\mathcal{B}\Sigma_1(<, AMOD)$ and $\mathcal{B}\Sigma_1(<, +1, AMOD)$ where “AMOD” is the set of *alphabetic modular predicates*. We obtain the decidability of separation for these classes (this is a new result for $BPol(AMT^+)$). However, we do not get a polynomial time upper bound: this is because AMT-separation is co-NP-complete (see [26]).

Important remark. Eilenberg’s theorem [7] connects some classes of regular languages (the “varieties of languages”) with *varieties of finite monoids*. It raised the hope to solve decision problems on languages (such as membership) by translating them in terms of monoids and solving the resulting purely algebraic questions – without referring to languages anymore. In particular, Margolis and Pin [11, 13] characterized the algebraic counterpart of $BPol(\mathcal{G})$ in Eilenberg’s correspondence (when \mathcal{G} is a variety) as the “*semidirect product*” $J * G$, where J is the variety of monoids corresponding to $\mathcal{B}\Sigma_1(<)$ and G is the one corresponding to \mathcal{G} . The new purely algebraic question is then: “decide membership of a monoid in $J * G$ ”. Tilson [35] developed an involved framework to reformulate membership in semidirect products in terms of categories, which was successfully exploited to handle $(J * G)$ -membership [8, 28].

Our results are completely independent from this algebraic approach. To clarify, we do use combinatorics on monoids. Yet, our motivations and techniques are disconnected from the theory of varieties of monoids, which is a distinct field. We avoid it *by choice*: while the above approach highlights an interesting connection between two fields, it is not necessarily desirable when looking back at our primary goal, understanding *classes of languages*. Indeed, a detour via varieties of monoids would obfuscate the intuition at the language level. Fortunately, this paper shows that this detour can be bypassed, while getting *stronger* results. First, our results are more general: they apply to *separation*, and not only membership. It is not clear at all that this can be obtained in the context of monoid varieties, as we rely strongly on the definition of $BPol$: we work with languages of the form $L_0 a_1 L_1 \cdots a_n L_n$, for $L_i \in \mathcal{G}$. Second, we can handle $BPol(\mathcal{G}^+)$, thus capturing the successor relation on the logical side. As far as we know, the only class of this kind captured by the above framework is $BPol(ST^+)$ (these are the well-known dot-depth one languages [30]). Third, using the algebraic approach via Eilenberg’s theorem requires *varieties* of languages as input classes. This, for example, excludes the class $BPol(MOD)$. This does not mean that this class cannot be handled by algebraic techniques: this was actually done by Straubing [31, 15], who rebuilt the whole theory to be able to handle such classes. In contrast, our result applies *uniformly* to classes of group languages, including MOD.

Organization of the paper. We present the objects that we investigate and terminology in Section 2. We introduce separation and the techniques that we use to handle it in Section 3. Finally, we present our results for $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation in Section 4. Due to space limitations, some proofs are only available in the full version of the paper [25].

2 Preliminaries

2.1 Words, regular languages and classes

We fix a finite *alphabet* A for the paper. As usual, A^* denotes the set of all finite words over A , including the empty word ε . We let $A^+ = A^* \setminus \{\varepsilon\}$. For $u, v \in A^*$, we let uv be the word obtained by concatenating u and v . A *language* is a subset of A^* . We denote the singleton language $\{u\}$ by u . We lift concatenation to languages: for $K, L \subseteq A^*$, we let $KL = \{uv \mid u \in K \text{ and } v \in L\}$. We shall consider *marked products*: given languages $L_0, \dots, L_n \subseteq A^*$, a marked product of L_0, \dots, L_n is a product of the form $L_0 a_1 L_1 \cdots a_n L_n$ where $a_1, \dots, a_n \in A$ (note that “ L_0 ” is a marked product: this is the case $n = 0$).

Regular languages. In the paper, we consider *regular* languages. A *nondeterministic finite automaton (NFA)* is a pair $\mathcal{A} = (Q, \delta)$ where Q is a finite set of states, and $\delta \subseteq Q \times A \times Q$ is a set of transitions. We now define the languages recognized by \mathcal{A} . Given $q, r \in Q$ and $w \in A^*$, we say that there exists a *run labeled by w from q to r* (in \mathcal{A}) if there exist $q_0, \dots, q_n \in Q$ and $a_1, \dots, a_n \in A$ such that $w = a_1 \cdots a_n$, $q_0 = q$, $q_n = r$ and $(q_{i-1}, a_i, q_i) \in \delta$ for every $1 \leq i \leq n$. Given two sets $I, F \subseteq Q$, we write $L_{\mathcal{A}}(I, F) \subseteq A^*$ for the language of all words $w \in A^*$ such that there exist $q \in I$, $r \in F$, and a run labeled by w from q to r in \mathcal{A} . We say that a language $L \subseteq A^*$ is *recognized* by \mathcal{A} if and only if there exist $I, F \subseteq Q$ such that $L = L_{\mathcal{A}}(I, F)$. The regular languages are those which can be recognized by an NFA.

We also use NFAs with ε -*transitions*. In such an NFA $\mathcal{A} = (Q, \delta)$, a transition may also be labeled by the empty word “ ε ” (that is, $\delta \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$). We use the standard semantics: an ε -transition can be taken without consuming an input letter. Note that unless otherwise specified, the NFAs that we consider are assumed to be *without* ε -transitions.

Classes. A *class* of languages is a set of languages. A *lattice of languages* is a class containing \emptyset and A^* and closed under both union and intersection. Moreover, a *Boolean algebra* is a lattice closed under complement. Finally, a class \mathcal{C} is *quotient-closed* when for all $L \in \mathcal{C}$ and all $v \in A^*$, the languages $v^{-1}L = \{w \in A^* \mid vw \in L\}$ and $Lv^{-1} = \{w \in A^* \mid wv \in L\}$ both belong to \mathcal{C} as well. A *positive prevariety* (resp. a *prevariety*) is a quotient-closed lattice (resp. a quotient-closed Boolean algebra) containing *regular languages only*.

Group languages. A *monoid* is a set M equipped with a multiplication $s, t \mapsto st$, which is associative and has a neutral element denoted by “ 1_M ”. Observe that A^* endowed with concatenation is a monoid (ε is the neutral element). It is well-known that a language L is regular if and only if it is *recognized* by a morphism $\alpha : A^* \rightarrow M$ into a *finite* monoid M , *i.e.*, there exists $F \subseteq M$ such that $L = \alpha^{-1}(F)$. We now restrict this definition: a monoid G is a *group* if every element $g \in G$ has an inverse $g^{-1} \in G$, *i.e.*, such that $gg^{-1} = g^{-1}g = 1_G$. A *group language* is a language recognized by a morphism into a *finite group*.

We consider classes \mathcal{G} that are group prevarieties (*i.e.*, containing group languages only). We let GR be the class of *all* group languages. Another important example is the class AMT of *alphabet modulo testable languages*. For every $w \in A^*$ and every $a \in A$, we write $\#_a(w) \in \mathbb{N}$ for the number of occurrences of “ a ” in w . The class AMT consists in all finite

Boolean combinations of languages $\{w \in A^* \mid \#_a(w) \equiv k \pmod m\}$ where $a \in A$ and $k, m \in \mathbb{N}$ are such that $k < m$. One may verify that these are exactly the languages recognized by commutative groups. Finally, we consider the class MOD, which consists in all finite Boolean combinations of languages $\{w \in A^* \mid |w| \equiv k \pmod m\}$ with $k, m \in \mathbb{N}$ such that $k < m$. Finally, we write ST for the trivial class $ST = \{\emptyset, A^*\}$. One may verify that GR, AMT, MOD and ST are all group prevarieties.

One may verify that $\{\varepsilon\}$ and A^+ are *not* group languages. This motivates the next definition: the *well-suited extension of a class \mathcal{C}* , denoted by \mathcal{C}^+ , consists of all languages of the form $L \cap A^+$ or $L \cup \{\varepsilon\}$ where $L \in \mathcal{C}$. The next lemma follows from the definition.

► **Lemma 1.** *Let \mathcal{C} be a prevariety. Then, \mathcal{C}^+ is a prevariety containing $\{\varepsilon\}$ and A^+ .*

2.2 Polynomial and Boolean closure

We investigate two operators that one may apply to a class \mathcal{C} . The *Boolean closure* of \mathcal{C} , written $Bool(\mathcal{C})$, is the least Boolean algebra containing \mathcal{C} . The *polynomial closure* of \mathcal{C} , denoted by $Pol(\mathcal{C})$, consists of all finite unions of marked products $L_0 a_1 L_1 \cdots a_n L_n$ where $L_0, \dots, L_n \in \mathcal{C}$ and $a_1, \dots, a_n \in A$. Finally, we write $BPol(\mathcal{C})$ for $Bool(Pol(\mathcal{C}))$. If \mathcal{C} is a prevariety, then $Pol(\mathcal{C})$ is a positive prevariety and $BPol(\mathcal{C})$ is a prevariety. Proving that $Pol(\mathcal{C})$ is closed under intersection is not immediate. It was shown by Arfi [2] (see also [14, 20]).

► **Theorem 2.** *If \mathcal{C} is a prevariety, $Pol(\mathcal{C})$ is a positive prevariety and $BPol(\mathcal{C})$ is a prevariety.*

The two operators Pol and $Bool$ induce standard classifications called concatenation hierarchies: for a prevariety \mathcal{C} , the *concatenation hierarchy of basis \mathcal{C}* is built from \mathcal{C} by alternatively applying the operators Pol and $Bool$. We are interested in $BPol(\mathcal{C})$, which is level *one* in the concatenation hierarchy of basis \mathcal{C} . We look at bases that are either a group prevariety \mathcal{G} or its well-suited extension \mathcal{G}^+ . Most of the prominent concatenation hierarchies in the literature use such bases. This is in part motivated by the logical characterization of concatenation hierarchies, due to Thomas [34]. We briefly recall it for the level one.

Consider a word $w = a_1 \cdots a_{|w|} \in A^*$. We view w as a linearly ordered set of $|w| + 2$ positions $\{0, 1, \dots, |w|, |w| + 1\}$ such that each position $1 \leq i \leq |w|$ carries the label $a_i \in A$ (on the other hand, 0 and $|w| + 1$ are artificial unlabeled leftmost and rightmost positions). We use first-order logic to describe properties of words: a sentence can quantify over the positions of a word and use a predetermined set of predicates to test properties of these positions. We also allow two constants “*min*” and “*max*” interpreted as the artificial unlabeled positions 0 and $|w| + 1$ in a given word w . A first-order sentence φ defines the language of all words satisfying the property stated by φ . We use several kinds of predicates. For each $a \in A$, we associate a unary predicate (also denoted by a), which selects the positions labeled by “ a ”. We also use two binary predicates: the (strict) linear order “ $<$ ” and the successor relation “ $+1$ ”. Finally, we associate a set of predicates $\mathbb{P}_{\mathcal{G}}$ to each group prevariety \mathcal{G} . Every $L \in \mathcal{G}$ yields a unary predicate P_L in $\mathbb{P}_{\mathcal{G}}$, which is interpreted as follows. Let $w = a_1 \cdots a_{|w|} \in A^*$. The unary predicate P_L selects all positions $i \in \{0, \dots, |w| + 1\}$ such that $i \neq 0$ and $a_1 \cdots a_{i-1} \in L$.

► **Example 3.** The sentence “ $\exists x \exists y (x < y) \wedge a(x) \wedge b(y)$ ” defines the language $A^* a A^* b A^*$. The sentence “ $\exists x \exists y a(x) \wedge c(y) \wedge (y + 1 = \text{max})$ ” defines $A^* a A^* c$. Finally, if $L = (AA)^* \in \text{MOD}$ (the words of even length), the sentence “ $\exists x a(x) \wedge P_L(x)$ ” defines the language $(AA)^* a A^*$.

The fragment of first-order logic containing exactly the Boolean combinations of existential first-order sentences is denoted by “ $\mathcal{B}\Sigma_1$ ”. Let \mathcal{G} be a group prevariety. We write $\mathcal{B}\Sigma_1(<, \mathbb{P}_{\mathcal{G}})$ for the class of all languages defined by a sentence of $\mathcal{B}\Sigma_1$ using only the label predicates,

the linear order “ $<$ ” and those in $\mathbb{P}_{\mathcal{G}}$. Moreover, we write $\mathcal{B}\Sigma_1(<, +1, \mathbb{P}_{\mathcal{G}})$ for the class of all languages defined by a sentence of $\mathcal{B}\Sigma_1$, which additionally allows the successor predicate “ $+1$ ”. The following proposition follows from the results of [20, 24].

► **Proposition 4.** *Let \mathcal{G} be a group prevariety. We have $BPol(\mathcal{G}) = \mathcal{B}\Sigma_1(<, \mathbb{P}_{\mathcal{G}})$ and $BPol(\mathcal{G}^+) = \mathcal{B}\Sigma_1(<, +1, \mathbb{P}_{\mathcal{G}})$.*

Key examples. The basis $ST = \{\emptyset, A^*\}$ yields the *Straubing-Thérien hierarchy* [29, 32] (hence the notation of this basis). Its level one is the class of piecewise testable languages [27]. Its well-suited extension ST^+ induces the *dot-depth hierarchy* [3]. In particular, $BPol(ST)$ and $BPol(ST^+)$ correspond to $\mathcal{B}\Sigma_1(<)$ and $\mathcal{B}\Sigma_1(<, +1)$, as all predicates in \mathbb{P}_{ST} are trivial. The hierarchies of bases MOD and MOD^+ are also prominent (see for example [5, 10, 36]). The classes $BPol(MOD)$ and $BPol(MOD^+)$ correspond to $\mathcal{B}\Sigma_1(<, MOD)$ and $\mathcal{B}\Sigma_1(<, +1, MOD)$ where “ MOD ” is the set of *modular predicates* (for all $r, q \in \mathbb{N}$ such that $r < q$, it contains a unary predicate $M_{r,q}$ selecting the positions i such that $i \equiv r \pmod{q}$). Similarly, $BPol(AMT)$ and $BPol(AMT^+)$ correspond to $\mathcal{B}\Sigma_1(<, AMOD)$ and $\mathcal{B}\Sigma_1(<, +1, AMOD)$ where “ $AMOD$ ” is the set of *alphabetic modular predicates* (for all $a \in A$ and $r, q \in \mathbb{N}$ such that $r < q$, it contains a unary predicate $M_{r,q}^a$ selecting the positions i such the that number of positions $j < i$ with label a is congruent to r modulo q). Finally, the group hierarchy, whose basis is GR is also prominent [11, 8], though its logical characterization is not standard.

Properties. We present a key ingredient [23, Lemma 3.6]. It describes a concatenation principle for the classes $BPol(\mathcal{C})$ based on the notion of “cover”. Given a language L , a cover of L is a *finite* set \mathbf{K} of languages satisfying $L \subseteq \bigcup_{K \in \mathbf{K}} K$. If \mathcal{D} is a class, a \mathcal{D} -cover of L is a cover \mathbf{K} of L such that $\mathbf{K} \subseteq \mathcal{D}$.

► **Proposition 5.** *Let \mathcal{C} be a prevariety, $n \in \mathbb{N}$, $L_0, \dots, L_n \in Pol(\mathcal{C})$ and $a_1, \dots, a_n \in A$. If \mathbf{H}_i is a $BPol(\mathcal{C})$ -cover of L_i for all $i \leq n$, then there is a $BPol(\mathcal{C})$ -cover \mathbf{K} of $L_0 a_1 L_1 \dots a_n L_n$ such that for all $K \in \mathbf{K}$, there exists $H_i \in \mathbf{H}_i$ for each $i \leq n$ satisfying $K \subseteq H_0 a_1 H_1 \dots a_n H_n$.*

For applying Proposition 5, we need a language $L_0 a_1 L_1 \dots a_n L_n$ with $L_0, \dots, L_n \in Pol(\mathcal{C})$. The next tailored statements build such languages when $\mathcal{C} = \mathcal{G}$ or \mathcal{G}^+ for a group prevariety \mathcal{G} . While simple, these results are central: this is the unique place where we use the fact that \mathcal{G} contains only *group languages*. Let $L \subseteq A^*$. With every word $w = a_1 \dots a_n \in A^*$, we associate the language $\uparrow_L w = L a_1 L \dots a_n L \subseteq A^*$ (we let $\uparrow_L \varepsilon = L$). We first present the statement for the case $\mathcal{C} = \mathcal{G}$, which can also be found in [4, Prop. 3.11].

► **Proposition 6.** *Let $H \subseteq A^*$ be a language and $L \subseteq A^*$ be a group language containing ε . There exists a cover \mathbf{K} of H such that every $K \in \mathbf{K}$ is of the form $K = \uparrow_L w$ for some $w \in H$.*

The next statement, useful for the case $\mathcal{C} = \mathcal{G}^+$, is a corollary of Proposition 6. Let $\mathcal{A} = (Q, \delta)$ be an NFA. Moreover, let $w, z \in A^*$. We say that z is a *left \mathcal{A} -loop* for w if for every $q, r \in Q$ such that $w \in L_{\mathcal{A}}(q, r)$, there exists $s \in Q$ such that $z \in L_{\mathcal{A}}(q, s) \cap L_{\mathcal{A}}(s, s)$ and $zw \in L_{\mathcal{A}}(s, r)$ (in particular, $zz^*zw \subseteq L_{\mathcal{A}}(q, r)$). Symmetrically, we say that z is a *right \mathcal{A} -loop* for w if for every $q, r \in Q$ such that $w \in L_{\mathcal{A}}(q, r)$, there exists $s \in Q$ such that $wz \in L_{\mathcal{A}}(q, s)$ and $z \in L_{\mathcal{A}}(s, s) \cap L_{\mathcal{A}}(s, r)$ (in particular, $wzz^*z \subseteq L_{\mathcal{A}}(q, r)$).

Now, given an arbitrary word $w \in A^*$, an *\mathcal{A} -guarded decomposition* of w is a tuple (w_1, \dots, w_{n+1}) for some $n \in \mathbb{N}$ where $w_1 \in A^*$ and $w_i \in A^+$ for $2 \leq i \leq n+1$, and such that $w = w_1 \dots w_{n+1}$ and, if $n \geq 1$, then for every i satisfying $1 \leq i \leq n$, there exists a *nonempty* word $z_i \in A^+$ which is a right \mathcal{A} -loop for w_i and a left \mathcal{A} -loop for w_{i+1} .

► **Proposition 7.** *Let $H \subseteq A^*$ be a language, \mathcal{A} be an NFA and $L \subseteq A^*$ be a group language containing ε . There exists a cover \mathbf{K} of H such that for each $K \in \mathbf{K}$, there exist a word $w \in H$ and an \mathcal{A} -guarded decomposition (w_1, \dots, w_{n+1}) of w for some $n \in \mathbb{N}$ such that $K = w_1 L \cdots w_n L w_{n+1}$ (if $n = 0$, then $K = \{w_1\}$).*

3 Separation framework

In order to investigate a given class \mathcal{C} , we rely on a generic decision problem that one may associate to it: \mathcal{C} -separation. We first define it and then present a variant, “tuple separation”, that we shall require as a proof ingredient.

3.1 The separation problem

Consider two languages $L_0, L_1 \subseteq A^*$. We say that a third language $K \subseteq A^*$ separates L_0 from L_1 when $L_0 \subseteq K$ and $K \cap L_1 = \emptyset$. Then, given an arbitrary class \mathcal{C} , we say that L_0 is \mathcal{C} -separable from L_1 when there exists $K \in \mathcal{C}$ that separates L_0 from L_1 . For every class \mathcal{C} , the \mathcal{C} -separation problem takes two regular languages L_0 and L_1 as input (in the paper, they are represented by NFAs) and asks whether L_0 is \mathcal{C} -separable from L_1 . We complete the definition with a useful result, which holds when \mathcal{C} is a positive prevariety.

► **Lemma 8.** *Let \mathcal{C} be a positive prevariety and $L_0, L_1, H_0, H_1 \subseteq A^*$. If L_0 is not \mathcal{C} -separable from L_1 and H_0 is not \mathcal{C} -separable from H_1 then $L_0 H_0$ is not \mathcal{C} -separable from $L_1 H_1$.*

In the paper, we look at \mathcal{C} -separation when $\mathcal{C} = BPol(\mathcal{G})$ or $BPol(\mathcal{G}^+)$ for a group prevariety \mathcal{G} . We prove that in these two cases, there are polynomial time (Turing) reductions to \mathcal{G} -separation. We now introduce terminology that we shall use to present the algorithms.

Framework. Consider a class \mathcal{C} and an NFA $\mathcal{A} = (Q, \delta)$. We associate a set $\mathcal{I}_{\mathcal{C}}[\mathcal{A}] \subseteq Q^4$: the *inseparable \mathcal{C} -quadruples* associated to \mathcal{A} . We define,

$$\mathcal{I}_{\mathcal{C}}[\mathcal{A}] = \{(q, r, s, t) \in Q^4 \mid L_{\mathcal{A}}(q, r) \text{ is \underline{not} } \mathcal{C}\text{-separable from } L_{\mathcal{A}}(s, t)\}.$$

The next easy result connects \mathcal{C} -separation to this set, for input languages given by NFAs.

► **Proposition 9.** *Let \mathcal{C} be a lattice. Consider an NFA $\mathcal{A} = (Q, \delta)$ and four sets of states $I_1, F_1, I_2, F_2 \subseteq Q$. The two following conditions are equivalent:*

1. $L_{\mathcal{A}}(I_1, F_1)$ is \mathcal{C} -separable from $L_{\mathcal{A}}(I_2, F_2)$.
2. $(I_1 \times F_1 \times I_2 \times F_2) \cap \mathcal{I}_{\mathcal{C}}[\mathcal{A}] = \emptyset$.

Clearly, given as input two regular languages recognized by NFAs, one may compute in polynomial time a single NFA recognizing both languages. Hence, Proposition 9 yields a polynomial time reduction from \mathcal{C} -separation to the problem of computing $\mathcal{I}_{\mathcal{C}}[\mathcal{A}]$ from an input NFA. Naturally, this does not necessarily mean that there exists a polynomial time algorithm for \mathcal{C} -separation: depending on \mathcal{C} , computing $\mathcal{I}_{\mathcal{C}}[\mathcal{A}]$ may or may not be costly.

We introduce a key definition for manipulating $\mathcal{I}_{\mathcal{C}}[\mathcal{A}]$, for an NFA $\mathcal{A} = (Q, \delta)$. Let $S \subseteq Q^4$ and \mathbf{K} be a finite set of languages. We say that \mathbf{K} is *separating for S* when for every $(q, r, s, t) \in Q^4$ and every $K \in \mathbf{K}$, if K intersects both $L_{\mathcal{A}}(q, r)$ and $L_{\mathcal{A}}(s, t)$, then $(q, r, s, t) \in S$. Then, $\mathcal{I}_{\mathcal{C}}[\mathcal{A}]$ is the smallest set of 4-tuples admitting a \mathcal{C} -cover of A^* which is separating for it.

► **Lemma 10.** *Let \mathcal{C} be a Boolean algebra and $\mathcal{A} = (Q, \delta)$ be an NFA. Then the following holds:*

- *There exists a \mathcal{C} -cover \mathbf{K} of A^* which is separating for $\mathcal{I}_{\mathcal{C}}[\mathcal{A}]$.*
- *Let $S \subseteq Q^4$. If there exists a \mathcal{C} -cover \mathbf{K} of A^* which is separating for S , then $\mathcal{I}_{\mathcal{C}}[\mathcal{A}] \subseteq S$.*

Controlled separation. We present additional terminology tailored to the classes built from a group prevariety. Consider two classes \mathcal{C} and \mathcal{D} (in practice, \mathcal{D} will be a group prevariety \mathcal{G} and \mathcal{C} will be either $BPol(\mathcal{G})$ or $BPol(\mathcal{G}^+)$). Let $L_0, L_1 \subseteq A^*$. We say that L_0 is \mathcal{C} -separable from L_1 under \mathcal{D} -control if there exists $H \in \mathcal{D}$ such that $\varepsilon \in H$ and $L_0 \cap H$ is \mathcal{C} -separable from $L_1 \cap H$. Given an NFA $\mathcal{A} = (Q, \delta)$, we associate a set $\mathcal{I}_{\mathcal{C}}[\mathcal{D}, \mathcal{A}] \subseteq Q^4$:

$$\mathcal{I}_{\mathcal{C}}[\mathcal{D}, \mathcal{A}] = \{(q, r, s, t) \in Q^4 \mid L_{\mathcal{A}}(q, r) \text{ is \underline{not} } \mathcal{C}\text{-separable from } L_{\mathcal{A}}(s, t) \text{ under } \mathcal{D}\text{-control}\}.$$

Clearly, we have $\mathcal{I}_{\mathcal{C}}[\mathcal{D}, \mathcal{A}] \subseteq \mathcal{I}_{\mathcal{C}}[\mathcal{A}]$. Let us connect this new definition to the notion of separating cover presented above. In this case as well, this will be useful in proof arguments.

► **Lemma 11.** *Let \mathcal{C} and \mathcal{D} be Boolean algebras such that $\mathcal{D} \subseteq \mathcal{C}$ and let $\mathcal{A} = (Q, \delta)$ be an NFA. The following properties hold:*

- *There exists $L \in \mathcal{D}$ with $\varepsilon \in L$, and a \mathcal{C} -cover \mathbf{K} of L which is separating for $\mathcal{I}_{\mathcal{C}}[\mathcal{D}, \mathcal{A}]$.*
- *Let $S \subseteq Q^4$. If there exist $L \in \mathcal{D}$ with $\varepsilon \in L$, and a \mathcal{C} -cover \mathbf{K} of L which is separating for S , then $\mathcal{I}_{\mathcal{C}}[\mathcal{D}, \mathcal{A}] \subseteq S$.*

This notion is only useful if $\{\varepsilon\} \notin \mathcal{D}$. If $\{\varepsilon\} \in \mathcal{D}$, then L_0 is \mathcal{C} -separable from L_1 under \mathcal{D} -control if and only if either $\varepsilon \notin L_0$ or $\varepsilon \notin L_1$. This is why the notion is designed for group prevarieties: if \mathcal{G} is such a class, then $\{\varepsilon\} \notin \mathcal{G}$. In this case, if $\mathcal{C} \in \{\mathcal{G}, \mathcal{G}^+\}$, then the set $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$ carries more information than $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$. This is useful for the computation: rather than computing $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ directly, our procedures first compute $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$. The proof is based on Propositions 5 and 6 (the latter requires \mathcal{G} to consist of group languages).

► **Proposition 12.** *Let \mathcal{G} be a group prevariety, let \mathcal{C} be a prevariety such that $\mathcal{G} \subseteq \mathcal{C}$ and let $\mathcal{A} = (Q, \delta)$ be an NFA. Then, $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ is the least set $S \subseteq Q^4$ that contains $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$ and satisfies the two following conditions:*

1. *For all $q, r, s, t \in Q$ and $a \in A$, if $(q, a, r), (s, a, t) \in \delta$, then $(q, r, s, t) \in S$.*
2. *For all $(q_1, r_1, s_1, t_1), (q_2, r_2, s_2, t_2) \in S$, if $r_1 = q_2$ and $t_1 = s_2$, then $(q_1, r_2, s_1, t_2) \in S$.*

Proof. Let $S \subseteq Q^4$ be the least set containing $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$ and satisfying both conditions. We prove that $S = \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$. For $S \subseteq \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$, since $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}] \subseteq \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ by definition, it suffices to prove that $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ satisfies both conditions in the proposition. First, consider $a \in A$ and $q, r, s, t \in Q$ such that $(q, a, r), (s, a, t) \in \delta$. We have $a \in L_{\mathcal{A}}(q, r)$ and $a \in L_{\mathcal{A}}(s, t)$. Hence, they are not $BPol(\mathcal{C})$ -separable and $(q, r, s, t) \in \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$. Now, let $(q_1, r_1, s_1, t_1), (q_2, r_2, s_2, t_2) \in \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ such that $r_1 = q_2$ and $t_1 = s_2$. For $i \in \{1, 2\}$, we know that $L_{\mathcal{A}}(q_i, r_i)$ is not $BPol(\mathcal{C})$ -separable from $L_{\mathcal{A}}(s_i, t_i)$. Since $BPol(\mathcal{C})$ is a prevariety by Theorem 2, it follows from Lemma 8 that $L_{\mathcal{A}}(q_1, r_1)L_{\mathcal{A}}(q_2, r_2)$ is not $BPol(\mathcal{C})$ separable from $L_{\mathcal{A}}(s_1, t_1)L_{\mathcal{A}}(s_2, t_2)$. Since $r_1 = q_2$ and $t_1 = s_2$, it is immediate that $L_{\mathcal{A}}(q_1, r_1)L_{\mathcal{A}}(q_2, r_2) \subseteq L_{\mathcal{A}}(q_1, r_2)$ and $L_{\mathcal{A}}(s_1, t_1)L_{\mathcal{A}}(s_2, t_2) \subseteq L_{\mathcal{A}}(s_1, t_2)$. Hence, $L_{\mathcal{A}}(q_1, r_2)$ is not $BPol(\mathcal{C})$ -separable from $L_{\mathcal{A}}(s_1, t_2)$ and we get $(q_1, r_2, s_1, t_2) \in \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ as desired.

We turn to the inclusion $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}] \subseteq S$. By Lemma 11, there exists $L \in \mathcal{G}$ such that $\varepsilon \in L$ and a $BPol(\mathcal{C})$ -cover \mathbf{V} of L which is separating for $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$. By hypothesis, L is a group language and $\varepsilon \in L$. Hence, Proposition 6 yields a cover \mathbf{P} of A^* such that every $P \in \mathbf{P}$ is of the form $P = \uparrow_L w_P$ for some word $w_P \in A^*$. Let $P \in \mathbf{P}$ and $a_1, \dots, a_n \in A$ be the letters such that $w_P = a_1 \cdots a_n$. We have $P = La_1L \cdots a_nL$ by definition (if $w_P = \varepsilon$, then $P = L$). By definition, $L \in \mathcal{G} \subseteq Pol(\mathcal{C})$. Hence, since \mathbf{V} is a $BPol(\mathcal{C})$ -cover of L , Proposition 5 yields a $BPol(\mathcal{C})$ -cover \mathbf{K}_P of P such that for every $K \in \mathbf{K}_P$, there are $V_0, \dots, V_n \in \mathbf{V}$ such that $K \subseteq V_0a_1V_1 \cdots a_nV_n$. We let $\mathbf{K} = \bigcup_{P \in \mathbf{P}} \mathbf{K}_P$. Since \mathbf{P} is a cover of A^* and \mathbf{K}_P is a $BPol(\mathcal{C})$ -cover of P for each $P \in \mathbf{P}$, \mathbf{K} is a $BPol(\mathcal{C})$ -cover of A^* . We show that \mathbf{K} is separating for S which implies that $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}] \subseteq S$ by Lemma 10.

Let $(q, r, s, t) \in Q^4$ and $K \in \mathbf{K}$ such that we have $x \in K \cap L_{\mathcal{A}}(q, r)$ and $y \in K \cap L_{\mathcal{A}}(s, t)$. We show that $(q, r, s, t) \in S$. We have $K \in \mathbf{K}_P$ for some $P \in \mathbf{P}$. Let $a_1, \dots, a_n \in A$ such that $w_P = a_1 \cdots a_n$. By definition, there are $V_0, \dots, V_n \in \mathbf{V}$ such that $K \subseteq V_0 a_1 V_1 \cdots a_n V_n$. Since $x, y \in K$, we get $x_i, y_i \in V_i$ for $0 \leq i \leq n$ such that $x = x_0 a_1 x_1 \cdots a_n x_n$ and $y = y_0 a_1 y_1 \cdots a_n y_n$. Since $x \in L_{\mathcal{A}}(q, r)$, we get $q_i, r_i \in Q$ for $0 \leq i \leq n$ such that $q_0 = q$, $r_n = r$, $x_i \in L_{\mathcal{A}}(q_i, r_i)$ for $0 \leq i \leq n$ and $(r_{i-1}, a_i, q_i) \in \delta$ for $1 \leq i \leq n$. Finally, since $y \in L_{\mathcal{A}}(s, t)$, we get $s_i, t_i \in Q$ for $0 \leq i \leq n$ such that $s_0 = s$, $t_n = t$, $y_i \in L_{\mathcal{A}}(s_i, t_i)$ for $0 \leq i \leq n$ and $(t_{i-1}, a_i, s_i) \in \delta$ for $1 \leq i \leq n$. Since S satisfies Condition 1 in the proposition, we get $(r_{i-1}, q_i, t_{i-1}, s_i) \in S$ for $1 \leq i \leq n$. Since $V_i \in \mathbf{V}$ which is separating for $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$ and $x_i, y_i \in V_i$, we also get $(q_i, r_i, q_i, t_i) \in \mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$ for $0 \leq i \leq n$. Thus, Condition 2 in the proposition yields $(q_0, r_0, s_n, t_n) \in S$, *i.e.* $(q, r, s, t) \in S$ as desired. \blacktriangleleft

Proposition 12 provides a least fixpoint algorithm for computing the set $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{A}]$ from $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$. Combined with Proposition 9, this yields a polynomial time reduction from $BPol(\mathcal{C})$ -separation to computing $\mathcal{I}_{BPol(\mathcal{C})}[\mathcal{G}, \mathcal{A}]$ from an NFA. We shall prove that when $\mathcal{C} \in \{\mathcal{G}, \mathcal{G}^+\}$, there are polynomial time reductions of the latter problem to \mathcal{G} -separation.

3.2 Tuple separation

This generalized variant of separation is taken from [18]. We shall use it as a proof ingredient: for every lattice \mathcal{C} , it is connected to the classical separation problem for $Bool(\mathcal{C})$. For every $n \geq 1$, we call “ n -tuple” a tuple of n languages (L_1, \dots, L_n) . In the sequel, given another language K , we shall write $(L_1, \dots, L_n) \cap K$ for the n -tuple $(L_1 \cap K, \dots, L_n \cap K)$. Let \mathcal{C} be a lattice, we use induction on n to define the \mathcal{C} -separable n -tuples:

- If $n = 1$, a 1-tuple (L_1) is \mathcal{C} -separable when $L_1 = \emptyset$.
- If $n \geq 2$, an n -tuple (L_1, \dots, L_n) is \mathcal{C} -separable when there exists $K \in \mathcal{C}$ such that $L_1 \subseteq K$ and $(L_2, \dots, L_n) \cap K$ is \mathcal{C} -separable. We call K a *separator* of (L_1, \dots, L_n) .

One may verify that classical separation is the special case $n = 2$. We generalize \mathcal{D} -controlled separation to this setting. For a class \mathcal{D} , we say that an n -tuple (L_1, \dots, L_n) is \mathcal{C} -separable under \mathcal{D} -control if there exists $H \in \mathcal{D}$ such that $\varepsilon \in H$ and $(L_1, \dots, L_n) \cap H$ is \mathcal{C} -separable.

We complete the definition with two simple properties of tuple separation. The second one is based on closure under quotients and generalizes Lemma 8.

► **Lemma 13.** *Let \mathcal{C} be a lattice and let $(L_1, \dots, L_n), (H_1, \dots, H_n)$ be two n -tuples. If $L_1 \cap \dots \cap L_n \neq \emptyset$, then (L_1, \dots, L_n) is not \mathcal{C} -separable. Moreover, if $L_i \subseteq H_i$ for every $i \leq n$ and (L_1, \dots, L_n) is not \mathcal{C} -separable, then (H_1, \dots, H_n) is not \mathcal{C} -separable either.*

► **Lemma 14.** *Let \mathcal{C} be a positive prevariety, $n \geq 1$ and let $(L_1, \dots, L_n), (H_1, \dots, H_n)$ be two n -tuples, which are not \mathcal{C} -separable. Then, $(L_1 H_1, \dots, L_n H_n)$ is not \mathcal{C} -separable either.*

A theorem of [18] connects tuple \mathcal{C} -separation for a lattice \mathcal{C} to $Bool(\mathcal{C})$ -separation: L_0 is $Bool(\mathcal{C})$ -separable from L_1 if and only if $(L_0, L_1)^p$ is \mathcal{C} -separable for some $p \geq 1$. Here, $(L_0, L_1)^p$ denotes the $2p$ -tuple obtained by concatenating p copies of (L_0, L_1) . For example, $(L_0, L_1)^3 = (L_0, L_1, L_0, L_1, L_0, L_1)$. We use a corollary applying to \mathcal{D} -controlled separation.

► **Corollary 15.** *Let \mathcal{C} and \mathcal{D} be two lattices such that $\mathcal{D} \subseteq \mathcal{C}$ and let $L_0, L_1 \subseteq A^*$. The following properties are equivalent:*

1. L_0 is $Bool(\mathcal{C})$ -separable from L_1 under \mathcal{D} -control.
2. There exists $p \geq 1$ such that $(L_0, L_1)^p$ is \mathcal{C} -separable under \mathcal{D} -control.

We only use the contrapositive of 1) \Rightarrow 2) in Corollary 15. We complete the presentation with two important lemmas about tuple separation for $Pol(\mathcal{D})$ and $Pol(\mathcal{D}^+)$. We use them to prove that tuples are not separable. Note that in practice, \mathcal{D} will be a group prevariety \mathcal{G} . Yet, the results are true regardless of this hypothesis.

► **Lemma 16.** *Let \mathcal{D} be a prevariety and (L_1, \dots, L_n) an n -tuple which is not $Pol(\mathcal{D})$ -separable under \mathcal{D} -control. Then, $(\{\varepsilon\}, L_1, \dots, L_n)$ is not $Pol(\mathcal{D})$ -separable.*

Proof. We prove the contrapositive. Assume that $(\{\varepsilon\}, L_1, \dots, L_n)$ is $Pol(\mathcal{D})$ -separable: we get $K \in Pol(\mathcal{D})$ such that $\varepsilon \in K$ and $(L_1, \dots, L_n) \cap K$ is $Pol(\mathcal{D})$ -separable. By definition, K is a finite union of marked product of languages in \mathcal{D} . Hence, since $\varepsilon \in K$, there exists a marked product involving a single language $H \in \mathcal{D}$ such that $\varepsilon \in H$ in the union defining K . In particular, $H \subseteq K$ and Lemma 13 implies that $(L_1, \dots, L_n) \cap H$ is $Pol(\mathcal{D})$ -separable. Since $H \in \mathcal{D}$ and $\varepsilon \in H$, it follows that (L_1, \dots, L_n) is $Pol(\mathcal{D})$ -separable under \mathcal{D} -control. ◀

► **Lemma 17.** *Let \mathcal{D} be a prevariety and $w \in A^+$. If (L_1, \dots, L_n) is not $Pol(\mathcal{D}^+)$ -separable under \mathcal{D} -control, then $(w^+, w^+L_1w^+, \dots, w^+L_nw^+)$ is not $Pol(\mathcal{D}^+)$ -separable.*

Proof. We prove the contrapositive. Assuming that $(w^+, w^+L_1w^+, \dots, w^+L_nw^+)$ is $Pol(\mathcal{D}^+)$ -separable, we show that (L_1, \dots, L_n) is $Pol(\mathcal{D}^+)$ -separable under \mathcal{D} -control. There exists $K \in Pol(\mathcal{D}^+)$ such that $w^+ \subseteq K$, and $(w^+L_1w^+, \dots, w^+L_nw^+) \cap K$ is $Pol(\mathcal{D}^+)$ -separable. By definition, K is a finite union of marked products $K_0a_1K_1 \cdots a_mK_m$ with $a_1, \dots, a_m \in A$ and $K_0, \dots, K_m \in \mathcal{D}^+$. Let $k \in \mathbb{N}$ such that $m \leq k$ for every product $K_0a_1K_1 \cdots a_mK_m$ in this union. Since $w^+ \subseteq K$, we have $w^{2(k+1)} \in K$. This yields a marked product $K_0a_1K_1 \cdots a_mK_m$ such that $w^{2(k+1)} \in K_0a_1K_1 \cdots a_mK_m \subseteq K$, $m \leq k$ and $K_0, \dots, K_m \in \mathcal{D}^+$. Therefore, we get $u_i \in K_i$ for each $i \leq m$ such that $w^{2(k+1)} = u_0a_1u_1 \cdots a_mu_m$. Moreover, since $m \leq k$, there exists $i \leq m$ such that ww is an infix of u_i . Thus, we get $x, y \in A^*$ and $\ell_1, \ell_2 \in \mathbb{N}$ such that $u_i = xw^{\ell_1}wy^{\ell_2}$, $u_0a_1u_1 \cdots a_ix = w^{\ell_1}$, $ya_{i+1}u_{i+1} \cdots a_mu_m = w^{\ell_2}$ and $\ell_1 + 2 + \ell_2 = 2(k+1)$.

By definition $K_i \in \mathcal{D}^+$ which yields $H \in \mathcal{D}$ such that either $K_i = H \cup \{\varepsilon\}$ or $K_i = H \cap A^+$. Hence, since $u_i \in K_i$ and $u_i \in A^+$ (recall that $w \in A^+$), we have $xw^{\ell_1}wy^{\ell_2} = u_i \in H$. Let $H' = (xw)^{-1}H(wy)^{-1}$. By closure under quotients, we have $H' \in \mathcal{D}$ and it is clear that $\varepsilon \in H'$ since $xw^{\ell_1}wy^{\ell_2} \in H$. Hence, it remains to prove that $(L_1, \dots, L_n) \cap H'$ is $Pol(\mathcal{D}^+)$ -separable. This will imply as desired that (L_1, \dots, L_n) is $Pol(\mathcal{D}^+)$ -separable under \mathcal{D} -control.

We know that $(w^+L_1w^+, \dots, w^+L_nw^+) \cap K$ is $Pol(\mathcal{D}^+)$ -separable. One may verify from the definitions that $w^{\ell_1+1}(L_j \cap H')w^{\ell_2+1} \subseteq w^+L_jw^+ \cap K$ for all $j \leq n$. Thus, Lemma 13 implies that $w^{\ell_1+1}(L_1 \cap H')w^{\ell_2+1}, \dots, w^{\ell_1+1}(L_n \cap H')w^{\ell_2+1}$ is $Pol(\mathcal{D}^+)$ -separable. Finally, since $(w^{\ell_1+1}, \dots, w^{\ell_1+1})$ and $(w^{\ell_2+1}, \dots, w^{\ell_2+1})$ are not $Pol(\mathcal{D}^+)$ -separable, it follows from Lemma 14 that $((L_1 \cap H'), \dots, (L_n \cap H'))$ is $Pol(\mathcal{D}^+)$ -separable as desired. ◀

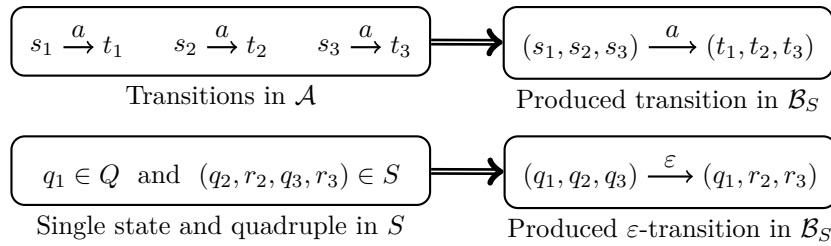
4 Separation Algorithms for $BPol(\mathcal{G})$ and $BPol(\mathcal{G}^+)$

For a group prevariety \mathcal{G} , we now consider $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation. We rely on the notions of Section 3: given an arbitrary NFA $\mathcal{A} = (Q, \delta)$, we present a generic characterization of the inseparable $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -quadruples under \mathcal{G} control associated to \mathcal{A} , *i.e.*, of the subsets $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$ and $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$ of Q^4 . Thanks to Proposition 12, this also yields characterizations of $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{A}]$ and of $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{A}]$, which in turn, in view of Proposition 9, yield reductions from both $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation to \mathcal{G} -separation. These polynomial time reductions are therefore *effective* when \mathcal{G} -separation is decidable.

4.1 Statements

Let \mathcal{G} be a group prevariety and let $\mathcal{A} = (Q, \delta)$ be an NFA. We present characterizations of $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$ and $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$. They follow the same pattern, but each of them depends on a specific function from 2^{Q^4} to 2^{Q^4} , which we first describe.

Characterization of $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$. We use a function $\tau_{\mathcal{A}, \mathcal{G}} : 2^{Q^4} \rightarrow 2^{Q^4}$. For $S \subseteq Q^4$, we define the set $\tau_{\mathcal{A}, \mathcal{G}}(S) \subseteq Q^4$. The definition is based on an auxiliary NFA $\mathcal{B}_S = (Q^3, \gamma_S)$ *with ε -transitions*, which depends on S . Its states are triples in Q^3 . The set $\gamma_S \subseteq Q^3 \times (A \cup \{\varepsilon\}) \times Q^3$ includes two kinds of transitions. First, given $a \in A$ and $s_1, s_2, s_3, t_1, t_2, t_3 \in Q$, we let $((s_1, s_2, s_3), a, (t_1, t_2, t_3)) \in \gamma_S$ if and only if $(s_1, a, t_1) \in \delta$, $(s_2, a, t_2) \in \delta$ and $(s_3, a, t_3) \in \delta$. Second, for every state $q_1 \in Q$ and every $(q_2, r_2, q_3, r_3) \in S$, we add the following ε -transition: $((q_1, q_2, q_3), \varepsilon, (q_1, r_2, r_3)) \in \gamma_S$. We represent this construction process graphically in Figure 1.



■ **Figure 1** Construction of the transitions in the auxiliary automaton \mathcal{B}_S .

► **Remark 18.** The NFA \mathcal{B}_S and its counterpart \mathcal{B}_S^+ (which we define below as a means to handle $BPol(\mathcal{G}^+)$) are the *only* NFAs with ε -transitions considered in the paper. In particular, the original input NFA \mathcal{A} is assumed to be *without* ε -transitions.

We are ready to define $\tau_{\mathcal{A}, \mathcal{G}}(S) \subseteq Q^4$. For every $(q, r, s, t) \in Q^4$, we let $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}(S)$ if and only if the two following conditions hold:

$$\begin{aligned} \{\varepsilon\} & \text{ is not } \mathcal{G}\text{-separable from } L_{\mathcal{B}_S}((s, q, s), (t, r, t)), \text{ and} \\ \{\varepsilon\} & \text{ is not } \mathcal{G}\text{-separable from } L_{\mathcal{B}_S}((q, s, q), (r, t, r)). \end{aligned} \quad (1)$$

A set $S \subseteq Q^4$ is $(BPol, *)$ -*sound* for \mathcal{G} and \mathcal{A} if it is a fixpoint for $\tau_{\mathcal{A}, \mathcal{G}}$, *i.e.* $\tau_{\mathcal{A}, \mathcal{G}}(S) = S$. We have the following simple lemma which can be verified from the definition. It states that $\tau_{\mathcal{A}, \mathcal{G}} : 2^{Q^4} \rightarrow 2^{Q^4}$ is *increasing* (for inclusion). In particular, this implies that it has a *greatest fixpoint*, *i.e.*, there is a *greatest* $(BPol, *)$ -*sound set*.

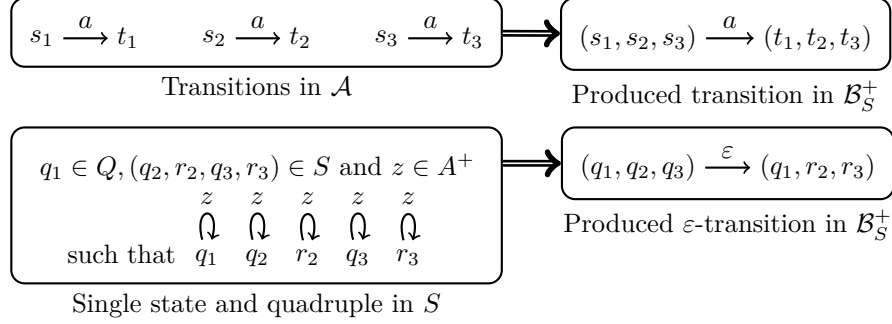
► **Lemma 19.** *Let \mathcal{G} be a group prevariety and let $\mathcal{A} = (Q, \delta)$ be an NFA. For every $S, S' \subseteq Q^4$, we have $S \subseteq S' \Rightarrow \tau_{\mathcal{A}, \mathcal{G}}(S) \subseteq \tau_{\mathcal{A}, \mathcal{G}}(S')$.*

We may now state the first key theorem of the paper. It applies to $BPol(\mathcal{G})$ -separation.

► **Theorem 20.** *Let \mathcal{G} be a group prevariety and $\mathcal{A} = (Q, \delta)$ an NFA. Then, $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$ is the greatest $(BPol, *)$ -*sound subset of Q^4 for \mathcal{G} and \mathcal{A} .**

Characterization of $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$. The characterization of $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$ is analogous. Roughly, the only difference is that we modify the definition of the auxiliary automaton \mathcal{B}_S . Let \mathcal{G} be a group prevariety and $\mathcal{A} = (Q, \delta)$ be an NFA. We define a new function $\tau_{\mathcal{A}, \mathcal{G}}^+ : 2^{Q^4} \rightarrow 2^{Q^4}$. For $S \subseteq Q^4$, we define $\tau_{\mathcal{A}, \mathcal{G}}^+(S) \subseteq Q^4$ using another auxiliary

NFA $\mathcal{B}_S^+ = (Q^3, \gamma_S^+)$ with ε -transitions. Its states are triples in Q^3 and $\gamma_S^+ \subseteq Q^3 \times (A \cup \{\varepsilon\}) \times Q^3$ contains two kinds of transitions. First, for $a \in A$ and $s_1, s_2, s_3, t_1, t_2, t_3 \in Q$, we let $((s_1, s_2, s_3), a, (t_1, t_2, t_3)) \in \gamma_S^+$ if and only if $(s_1, a, t_1) \in \delta$, $(s_2, a, t_2) \in \delta$ and $(s_3, a, t_3) \in \delta$. Second, for all $q_1 \in Q$ and all $(q_2, r_2, q_3, r_3) \in S$, if $A^+ \cap L_{\mathcal{A}}(q_1, q_1) \cap L_{\mathcal{A}}(q_2, q_2) \cap L_{\mathcal{A}}(q_3, q_3) \cap L_{\mathcal{A}}(r_2, r_2) \cap L_{\mathcal{A}}(r_3, r_3) \neq \emptyset$, then we add the following ε -transition: $((q_1, q_2, q_3), \varepsilon, (q_1, r_2, r_3)) \in \gamma_S^+$. We represent this construction in Figure 2.



■ **Figure 2** Construction of the transitions in the auxiliary automaton \mathcal{B}_S^+ .

We are ready to define $\tau_{\mathcal{A}, \mathcal{G}}^+(S) \subseteq Q^4$. For every $(q, r, s, t) \in Q^4$, we let $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}^+(S)$ if and only if the two following conditions hold:

$$\begin{aligned} \{\varepsilon\} \text{ is not } \mathcal{G}\text{-separable from } L_{\mathcal{B}_S^+}((s, q, s), (t, r, t)), \text{ and} \\ \{\varepsilon\} \text{ is not } \mathcal{G}\text{-separable from } L_{\mathcal{B}_S^+}((q, s, q), (r, t, r)). \end{aligned} \quad (2)$$

A set $S \subseteq Q^4$ is $(BPol, +)$ -sound for \mathcal{G} and \mathcal{A} if it is a fixpoint for $\tau_{\mathcal{A}, \mathcal{G}}^+$, i.e. $\tau_{\mathcal{A}, \mathcal{G}}^+(S) = S$. The following monotonicity lemma implies that there is a *greatest* $(BPol, +)$ -sound set.

► **Lemma 21.** *Let \mathcal{G} be a group prevariety and $\mathcal{A} = (Q, \delta)$ an NFA. For every $S, S' \subseteq Q^4$, we have $S \subseteq S' \Rightarrow \tau_{\mathcal{A}, \mathcal{G}}^+(S) \subseteq \tau_{\mathcal{A}, \mathcal{G}}^+(S')$.*

We may now state our second key theorem. It applies to $BPol(\mathcal{G}^+)$ -separation.

► **Theorem 22.** *Let \mathcal{G} be a group prevariety and $\mathcal{A} = (Q, \delta)$ an NFA. Then, $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$ is the greatest $(BPol, +)$ -sound subset of Q^4 for \mathcal{G} and \mathcal{A} .*

Let us discuss the consequences of Theorems 20 and 22. Since \mathcal{B}_S and \mathcal{B}_S^+ can be computed from \mathcal{A} and S , one can compute $\tau_{\mathcal{A}, \mathcal{G}}(S)$ and $\tau_{\mathcal{A}, \mathcal{G}}^+(S)$ from S provided that \mathcal{G} -separation is decidable. Hence, if \mathcal{G} -separation is decidable, Theorem 20 (resp. Theorem 22) yields a *greatest* fixpoint procedure for computing $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$ (resp. $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$). Indeed, consider the sequence of subsets defined by $S_0 = Q^4$, and $S_n = \tau_{\mathcal{A}, \mathcal{G}}(S_{n-1})$ for $n \geq 1$. By definition, computing S_n from S_{n-1} boils down to deciding \mathcal{G} -separation. Since $\tau_{\mathcal{A}, \mathcal{G}}$ is increasing by Lemma 19, we get a decreasing sequence $Q^4 = S_0 \supseteq S_1 \supseteq S_2 \dots$. Moreover, since Q^4 is finite, this sequence stabilizes at some point: there exists $n \in \mathbb{N}$ such that $S_n = S_j$ for all $j \geq n$. One may verify that S_n is the greatest $(BPol, *)$ -sound subset of Q^4 . By Theorem 20, it follows that $S_n = \mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$. Likewise, the sequence T_n defined by $T_0 = Q^4$ and $T_n = \tau_{\mathcal{A}, \mathcal{G}}^+(T_{n-1})$ is computable when \mathcal{G} -separation is decidable, and, since it is decreasing, it stabilizes. By Theorem 22, its stabilization value is $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$.

By Proposition 12, $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{A}]$ (resp. $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{A}]$) can be computed from $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$ (resp. $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$) via a *least* fixpoint procedure. Altogether, by Proposition 9, we get reductions from $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation to \mathcal{G} -separation. One may verify that these are polynomial time reductions (we mean “reduction” in the Turing sense: $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation can be decided in polynomial time using an oracle for \mathcal{G} -separation).

Now, it is known that separation can be decided in polynomial time for the classes ST, MOD and GR (this is trivial for ST, see [26] for MOD and GR). Hence, we obtain from Theorem 20 that separation is decidable in polynomial time for $BPol(\text{ST})$ (i.e., $\mathcal{B}\Sigma_1(<)$), $BPol(\text{MOD})$ (i.e., $\mathcal{B}\Sigma_1(<, \text{MOD})$) and $BPol(\text{GR})$. This was well-know for $BPol(\text{ST})$ (the class of piecewise testable languages, see [6, 17]). For the other two, decidability was known [36, 21] but not the polynomial time upper bound. Using Theorem 22, we also obtain that separation is decidable in polynomial time for $BPol(\text{ST}^+)$ (i.e., the languages of dot-depth one or equivalently $\mathcal{B}\Sigma_1(<, +1)$), $BPol(\text{MOD}^+)$ (i.e., $\mathcal{B}\Sigma_1(<, +1, \text{MOD})$) and $BPol(\text{GR}^+)$. Decidability was already known for $BPol(\text{ST}^+)$ and $BPol(\text{MOD}^+)$: the results can be obtained indirectly by reduction to $BPol(\text{ST})$ -separation using transfer theorems [22, 16]. Yet, the polynomial time upper bounds are new as the transfer theorems have a built-in exponential blow-up. Moreover, decidability of separation is a new result for $BPol(\text{GR}^+)$.

Finally, the statement applies to $BPol(\text{AMT})$ and $BPol(\text{AMT}^+)$ (i.e., $\mathcal{B}\Sigma_1(<, \text{AMOD})$ and $\mathcal{B}\Sigma_1(<, +1, \text{AMOD})$). This is a new result for $BPol(\text{AMT}^+)$. Yet, since AMT-separation is co-NP-complete when the alphabet is part of the input [26] (the problem being in P for a fixed alphabet), the complexity analysis is not entirely immediate. However, one may verify that the procedures yield co-NP algorithms for both $BPol(\text{AMT})$ - and $BPol(\text{AMT}^+)$ -separation. We summarize the upper bounds in Figure 3.

Input class \mathcal{G}	ST	MOD	AMT	GR
$BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation	P	P	co-NP	P

■ **Figure 3** Complexity of separation (for input languages represented by NFAs).

4.2 Proof of Theorem 20

We now concentrate on the proof of Theorem 20. The key ingredients in this argument are Proposition 6 and Lemma 16. The proof of Theorem 22 is available in the appendix. It is based on similar ideas. Roughly, we replace Proposition 6 and Lemma 16 (which are tailored to classes $BPol(\mathcal{G})$) by their counterparts for $BPol(\mathcal{G}^+)$: Proposition 7 and Lemma 17. However, note that proving Theorem 22 is technically more involved as manipulating the automaton \mathcal{B}_S^+ in the definition of $\tau_{\mathcal{A}, \mathcal{G}}^+$ requires more work.

We fix a group prevariety \mathcal{G} and an NFA $\mathcal{A} = (Q, \delta)$. Let $S \subseteq Q^4$ be the greatest $(BPol, *)$ -sound subset for \mathcal{G} and \mathcal{A} . We prove that $S = \mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$.

First part: $S \subseteq \mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$. We use *tuple separation* and Lemma 16. Let us start with some terminology. For every $n \geq 1$ and $(q_1, r_1, q_2, r_2) \in Q^4$, we associate an n -tuple of languages, written $T_n(q_1, r_1, q_2, r_2)$. We use induction on n and tuple concatenation to present the definition. If $n = 1$ then, $T_1(q_1, r_1, q_2, r_2) = (L_{\mathcal{A}}(q_2, r_2))$. If $n > 1$, then,

$$T_n(q_1, r_1, q_2, r_2) = \begin{cases} (L_{\mathcal{A}}(q_2, r_2)) \cdot T_{n-1}(q_1, r_1, q_2, r_2) & \text{if } n \text{ is odd} \\ (L_{\mathcal{A}}(q_1, r_1)) \cdot T_{n-1}(q_1, r_1, q_2, r_2) & \text{if } n \text{ is even.} \end{cases}$$

For example, we have $T_3(q_1, r_1, q_2, r_2) = (L_{\mathcal{A}}(q_2, r_2), L_{\mathcal{A}}(q_1, r_1), L_{\mathcal{A}}(q_2, r_2))$.

► **Proposition 23.** *For every $n \geq 1$ and $(q_1, r_1, q_2, r_2) \in S$, the n -tuple $T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable under \mathcal{G} -control.*

By definition, Proposition 23 implies that for all $p \geq 1$ and $(q_1, r_1, q_2, r_2) \in S$, the $2p$ -tuple $(L_{\mathcal{A}}(q_1, r_1), L_{\mathcal{A}}(q_2, r_2))^p$ is not $Pol(\mathcal{G})$ -separable under \mathcal{G} -control. By Corollary 15, it follows that $L_{\mathcal{A}}(q_1, r_1)$ is not $BPol(\mathcal{G})$ -separable from $L_{\mathcal{A}}(q_2, r_2)$ under \mathcal{G} -control, *i.e.*, that $(q_1, r_1, q_2, r_2) \in \mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$. We get $S \subseteq \mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$ as desired.

We prove Proposition 23 by induction on n . We fix $n \geq 1$ for the proof. In order to exploit the hypothesis that S is $(BPol, *)$ -sound, we need a property of the NFA $\mathcal{B}_S = (Q^3, \gamma_S)$ used to define $\tau_{\mathcal{A}, \mathcal{G}}$. When $n \geq 2$, this is where we use induction on n and Lemma 16.

► **Lemma 24.** *Let $(s_1, s_2, s_3), (t_1, t_2, t_3) \in Q^3$ and $w \in L_{\mathcal{B}_S}((s_1, s_2, s_3), (t_1, t_2, t_3))$. Then, $w \in L_{\mathcal{A}}(s_1, t_1)$ and, if $n \geq 2$, the n -tuple $(\{w\}) \cdot T_{n-1}(s_2, t_2, s_3, t_3)$ is not $Pol(\mathcal{G})$ -separable.*

Proof. Since $w \in L_{\mathcal{B}_S}((s_1, s_2, s_3), (t_1, t_2, t_3))$, there exists a run labeled by w from (s_1, s_2, s_3) to (t_1, t_2, t_3) in \mathcal{B}_S . We use a sub-induction on the number of transitions involved in that run. First, assume that no transitions are used: we have $w = \varepsilon$ and $(s_1, s_2, s_3) = (t_1, t_2, t_3)$. Clearly, $\varepsilon \in L_{\mathcal{A}}(s_1, s_1)$ and, if $n \geq 2$, the n -tuple $(\{\varepsilon\}) \cdot T_{n-1}(s_2, s_2, s_3, s_3)$ is not $Pol(\mathcal{G})$ -separable by Lemma 13 since $\varepsilon \in L_{\mathcal{A}}(s_2, s_2) \cap L_{\mathcal{A}}(s_3, s_3)$. We now assume that at least one transition is used and consider the last one: we have $(q_1, q_2, q_3) \in Q^3$, $w' \in A^*$ and $x \in A \cup \{\varepsilon\}$ such that $w = w'x$, $w' \in L_{\mathcal{B}_S}((s_1, s_2, s_3), (q_1, q_2, q_3))$ and $((q_1, q_2, q_3), x, (t_1, t_2, t_3)) \in \gamma_S$. By induction, we have $w' \in L_{\mathcal{A}}(s_1, q_1)$ and, if $n \geq 2$, the n -tuple $(\{w'\}) \cdot T_{n-1}(s_2, q_2, s_3, q_3)$ is not $Pol(\mathcal{G})$ -separable. We prove that $x \in L_{\mathcal{A}}(q_1, t_1)$ and, if $n \geq 2$, the n -tuple $(\{x\}) \cdot T_{n-1}(q_2, t_2, q_3, t_3)$ is not $Pol(\mathcal{G})$ -separable. It will then be immediate that $w = w'x \in L_{\mathcal{A}}(s_1, t_1)$ and, if $n \geq 2$, Lemma 14 implies that $(\{w\}) \cdot T_{n-1}(s_2, t_2, s_3, t_3)$ is not $Pol(\mathcal{G})$ -separable.

We consider two cases depending on whether $x \in A$ or $x = \varepsilon$. First, if $x = a \in A$, then $(q_i, a, t_i) \in \delta$ for $i = \{1, 2, 3\}$. Clearly, this implies that $a \in L_{\mathcal{A}}(q_1, t_1)$ and, if $n \geq 2$, then $(\{a\}) \cdot T_{n-1}(q_2, t_2, q_3, t_3)$ is not $Pol(\mathcal{G})$ -separable by Lemma 13 since $a \in L_{\mathcal{A}}(q_2, t_2) \cap L_{\mathcal{A}}(q_3, t_3)$. Assume now that $x = \varepsilon$: we are dealing with an ε -transition. By definition of γ_S , we have $q_1 = t_1$ and $(q_2, t_2, q_3, t_3) \in S$. The former yields $\varepsilon \in L_{\mathcal{A}}(q_1, t_1)$. Moreover, if $n \geq 2$, since $(q_2, t_2, q_3, t_3) \in S$, it follows from induction on n in Proposition 23 that the $(n-1)$ -tuple $T_{n-1}(q_2, t_2, q_3, t_3)$ is not $Pol(\mathcal{G})$ -separable under \mathcal{G} -control. Combined with Lemma 16, this yields that $(\{\varepsilon\}) \cdot T_{n-1}(q_2, t_2, q_3, t_3)$ is not $Pol(\mathcal{G})$ -separable, as desired. ◀

We may now complete the proof of Proposition 23. By symmetry, we only treat the case when n is odd and leave the case when it is even to the reader. Let $(q_1, r_1, q_2, r_2) \in S$, we have to prove that $T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable under \mathcal{G} -control. Hence, we fix $H \in \mathcal{G}$ such that $\varepsilon \in H$ and prove $H \cap T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable. Since S is $(BPol, *)$ -sound, we have $\tau_{\mathcal{A}, \mathcal{G}}(S) = S$, which implies that $(q_1, r_1, q_2, r_2) \in \tau_{\mathcal{A}, \mathcal{G}}(S)$. Hence, it follows from (1) that $\{\varepsilon\}$ is not \mathcal{G} -separable from $L_{\mathcal{B}_S}((q_2, q_1, q_2), (r_2, r_1, r_2))$. Since $H \in \mathcal{G}$ and $\varepsilon \in H$, we get a word $w \in H \cap L_{\mathcal{B}_S}((q_2, q_1, q_2), (r_2, r_1, r_2))$. By Lemma 24, we have $w \in H \cap L_{\mathcal{A}}(q_2, r_2)$. This completes the proof when $n = 1$. Indeed, in that case we have $T_1(q_1, r_1, q_2, r_2) = (L_{\mathcal{A}}(q_2, r_2))$ and since $H \cap L_{\mathcal{A}}(q_2, r_2) \neq \emptyset$, it follows that $H \cap T_1(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable, as desired. If $n \geq 2$, then Lemma 24 also implies that $(\{w\}) \cdot T_{n-1}(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable. Since $w \in H \cap L_{\mathcal{A}}(q_2, r_2)$, Lemma 13 yields that $(H \cap L_{\mathcal{A}}(q_2, r_2)) \cdot T_{n-1}(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable. Thus, since $H \in \mathcal{G} \subseteq Pol(\mathcal{G})$, one may verify that the n -tuple $(H \cap L_{\mathcal{A}}(q_2, r_2)) \cdot (H \cap T_{n-1}(q_1, r_1, q_2, r_2))$ is not $Pol(\mathcal{G})$ -separable. By definition, this exactly says that $H \cap T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G})$ -separable, completing the proof.

Second part: $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}] \subseteq S$. In the sequel, we say that an arbitrary set $R \subseteq Q^4$ is *good* if there exists $L \in \mathcal{G}$ such $\varepsilon \in L$ and a $BPol(\mathcal{G})$ -cover \mathbf{K} of L which is separating for R .

► **Proposition 25.** *Let $R \subseteq Q^4$. If R is good, then $\tau_{\mathcal{A}, \mathcal{G}}(R)$ is good as well.*

We use Proposition 25 to complete the proof. Let $S_0 = Q^4$ and $S_i = \tau_{\mathcal{A}, \mathcal{G}}(S_{i-1})$ for $i \geq 1$. By Lemma 19, we have $S_0 \supseteq S_1 \subseteq S_2 \supseteq \dots$ and there is $n \in \mathbb{N}$ such that S_n is the greatest $(BPol, *)$ -sound subset for \mathcal{G} and \mathcal{A} , *i.e.*, such that $S_n = S$. Since S_0 is good ($\{A^*\}$ is a $BPol(\mathcal{G})$ -cover of $A^* \in \mathcal{G}$ which is separating for $S_0 = Q^4$), Proposition 25 implies that S_i is good for all $i \in \mathbb{N}$. Thus, $S = S_n$ is good. We get $L \in \mathcal{G}$ such that $\varepsilon \in L$ and a $BPol(\mathcal{G})$ -cover \mathbf{K} of L which is separating for S . Lemma 11 then yields $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}] \subseteq S$ as desired.

► **Remark 26.** The proof of Proposition 25 actually provides a construction for building $L \in \mathcal{G}$ such that $\varepsilon \in L$ and a $BPol(\mathcal{G})$ -cover \mathbf{K} of L which is separating for S (yet, this involves building separators in \mathcal{G} , see Lemma 27). As we have now established that $S = \mathcal{I}_{BPol(\mathcal{G})}[\mathcal{G}, \mathcal{A}]$, one may then follow the proof of Proposition 12 to build a $BPol(\mathcal{G})$ -cover \mathbf{H} of A^* which is separating for $\mathcal{I}_{BPol(\mathcal{G})}[\mathcal{A}]$. Finally, \mathbf{H} encodes separators for all pairs of languages recognized by \mathcal{A} which are $BPol(\mathcal{G})$ -separable (roughly, this is the proof of Lemma 10). Altogether, we get a way to build separators in $BPol(\mathcal{G})$, when they exist.

We now prove Proposition 25. Let $R \subseteq Q^4$ be good. We have to build $L \in \mathcal{G}$ with $\varepsilon \in L$ and a $BPol(\mathcal{G})$ -cover \mathbf{K} of L which is separating for $\tau_{\mathcal{A}, \mathcal{G}}(R)$ (which will prove that $\tau_{\mathcal{A}, \mathcal{G}}(R)$ is good as well). We first build L (this part is independent from our hypothesis on R).

► **Lemma 27.** *There exists $L \in \mathcal{G}$ such that $\varepsilon \in L$ and for every $(q, r, s, t) \in Q^4$, if $L_{\mathcal{B}_R}((q, s, q), (r, t, r)) \cap L \neq \emptyset$ and $L_{\mathcal{B}_R}((s, q, s), (t, r, t)) \cap L \neq \emptyset$, then $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}(R)$.*

Proof. Let \mathbf{H} be the *finite* set of all languages recognized by \mathcal{B}_R such that $\{\varepsilon\}$ is \mathcal{G} -separable from H . For every $H \in \mathbf{H}$, there exists $L_H \in \mathcal{G}$ such that $\varepsilon \in L_H$ and $L_H \cap H = \emptyset$. We define $L = \bigcap_{H \in \mathbf{H}} L_H \in \mathcal{G}$. It is clear that $\varepsilon \in L$. Moreover, given $(q, r, s, t) \in Q^4$, if $L_{\mathcal{B}_R}((q, s, q), (r, t, r)) \cap L \neq \emptyset$ and $L_{\mathcal{B}_R}((s, q, s), (t, r, t)) \cap L \neq \emptyset$, it follows from the definition of L that $\{\varepsilon\}$ is not \mathcal{G} -separable from both $L_{\mathcal{B}_R}((q, s, q), (r, t, r))$ and $L_{\mathcal{B}_R}((s, q, s), (t, r, t))$. It follows from (1) in the definition of $\tau_{\mathcal{A}, \mathcal{G}}$ that $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}(R)$. ◀

We fix $L \in \mathcal{G}$ as described in Lemma 27 for the remainder of the proof. We now build the $BPol(\mathcal{G})$ -cover \mathbf{K} of L using the hypothesis that R is good and Proposition 6.

► **Lemma 28.** *For all $(q, r) \in Q^2$, there is $H_{q,r} \in BPol(\mathcal{G})$ such that $L_{\mathcal{A}}(q, r) \cap L \subseteq H_{q,r}$ and for all pairs $(s, t) \in Q^2$, if $L_{\mathcal{A}}(s, t) \cap H_{q,r} \neq \emptyset$ then $L_{\mathcal{B}_R}((q, s, q), (r, t, r)) \cap L \neq \emptyset$.*

Proof. Since R is good, there are $U \in \mathcal{G}$ such that $\varepsilon \in U$ and a $BPol(\mathcal{G})$ -cover \mathbf{V} of U which is separating for R . We use them to build $H_{q,r}$. Since U is a group language and $\varepsilon \in U$, Proposition 6 yields a cover \mathbf{P} of $L_{\mathcal{A}}(q, r) \cap L$ such that every $P \in \mathbf{P}$ is of the form $P = \uparrow_U w_P$ where $w_P \in L_{\mathcal{A}}(q, r) \cap L$. For every $P \in \mathbf{P}$, we build a $BPol(\mathcal{G})$ -cover \mathbf{K}_P of P . Let $a_1, \dots, a_n \in A$ be the letters such that $w_P = a_1 \dots a_n$. We have $P = U a_1 U \dots a_n U$. Since $U \in \mathcal{G} \subseteq Pol(\mathcal{G})$ and \mathbf{V} is a $BPol(\mathcal{G})$ -cover of U , Proposition 5 yields a $BPol(\mathcal{G})$ -cover \mathbf{K}_P of P such that for every $K \in \mathbf{K}_P$, there exist $V_0, \dots, V_n \in \mathbf{V}$ satisfying $K \subseteq V_0 a_1 V_1 \dots a_n V_n$. We define $H_{q,r}$ as the union of all languages K such that $K \in \mathbf{K}_P$ for some $P \in \mathbf{P}$ and $L_{\mathcal{A}}(q, r) \cap K \neq \emptyset$. Clearly, $H_{q,r} \in BPol(\mathcal{G})$. Moreover, since \mathbf{P} is a cover of $L_{\mathcal{A}}(q, r) \cap L$, and \mathbf{K}_P is a cover of P for each $P \in \mathbf{P}$, it is clear that $L_{\mathcal{A}}(q, r) \cap L \subseteq H_{q,r}$. We now fix $(s, t) \in Q^2$ such that $L_{\mathcal{A}}(s, t) \cap H_{q,r} \neq \emptyset$ and show that $L_{\mathcal{B}_R}((q, s, q), (r, t, r)) \cap L \neq \emptyset$. By definition of $H_{q,r}$, we get $P \in \mathbf{P}$ and $K \in \mathbf{K}_P$ such that $L_{\mathcal{A}}(q, r) \cap K \neq \emptyset$ and $L_{\mathcal{A}}(s, t) \cap K \neq \emptyset$. By definition, $P = \uparrow_U w_P$ with $w_P \in L_{\mathcal{A}}(q, r) \cap L$. Hence, it suffices to prove that $w_P \in L_{\mathcal{B}_R}((q, s, q), (r, t, r))$.

We fix $x \in L_{\mathcal{A}}(s, t) \cap K$ and $y \in L_{\mathcal{A}}(q, r) \cap K$. Recall that $w_P = a_1 \cdots a_n$ (if $n = 0$, then $w_P = \varepsilon$). Since $w_P \in L_{\mathcal{A}}(q, r)$, we may consider the corresponding run in \mathcal{A} : we get $p_0, \dots, p_n \in Q$ such that $p_0 = q$, $p_n = r$ and $(p_{i-1}, a_i, p_i) \in \delta$ for $1 \leq i \leq n$. Moreover, since $K \in \mathbf{K}_P$ and $w_P = a_1 \cdots a_n$, we have $K \subseteq V_0 a_1 V_1 \cdots a_n V_n$ for $V_0, \dots, V_n \in \mathbf{V}$ (if $n = 0$, then $K \subseteq V_0$). Since $x, y \in K$, we get $x_i, y_i \in V_i$ for $0 \leq i \leq n$ such that $x = x_0 a_1 x_1 \cdots a_n x_n$ and $y = y_0 a_1 y_1 \cdots a_n y_n$. Since $x \in L_{\mathcal{A}}(s, t)$, we get $s_0, t_0, \dots, s_n, t_n \in Q$ such that $s_0 = s$, $t_n = t$, $x_i \in L_{\mathcal{A}}(s_i, t_i)$ for $0 \leq i \leq n$, and $(t_{i-1}, a_i, s_i) \in \delta$ for $1 \leq i \leq n$. Symmetrically, since $y \in L_{\mathcal{A}}(q, r)$, we get $q_0, r_0, \dots, q_n, r_n \in Q$ such that $q_0 = q$, $r_n = r$, $y_i \in L_{\mathcal{A}}(q_i, r_i)$ for $0 \leq i \leq n$, and $(r_{i-1}, a_i, q_i) \in \delta$ for $1 \leq i \leq n$. By definition of γ_R , it is immediate that $((p_{i-1}, t_{i-1}, r_{i-1}), a_i, (p_i, s_i, q_i)) \in \gamma_R$ for $1 \leq i \leq n$. Since $V_i \in \mathbf{V}$ and \mathbf{V} is separating for R , the fact that $x_i, y_i \in V_i$ implies that $(s_i, t_i, q_i, r_i) \in R$ for $0 \leq i \leq n$. Hence, $((p_i, s_i, q_i), \varepsilon, (p_i, t_i, r_i)) \in \gamma_R$ by definition. Thus, we get a run labeled by w_P from (p_0, s_0, q_0) to (p_n, t_n, r_n) in \mathcal{B}_R , *i.e.*, $w_P \in L_{\mathcal{B}_R}((q, s, q), (r, t, r))$ as desired. \blacktriangleleft

We may now build \mathbf{K} . Let $\mathbf{H} = \{H_{q,r} \mid (q, r) \in Q^2\}$. Consider the following equivalence \sim defined on L : given $u, v \in L$, we let $u \sim v$ if and only if $u \in H_{q,r} \Leftrightarrow v \in H_{q,r}$ for every $(q, r) \in Q^2$. We let \mathbf{K} as the partition of L into \sim -classes. Clearly, each $K \in \mathbf{K}$ is a Boolean combination involving the languages in \mathbf{H} (which belong to $BPol(\mathcal{G})$) and $L \in \mathcal{G}$. Hence, \mathbf{K} is a $BPol(\mathcal{G})$ -cover of L . We now prove that it is separating for $\tau_{\mathcal{A}, \mathcal{G}}(R)$. Let $q, r, s, t \in Q$ and $K \in \mathbf{K}$ such that there are $u \in L_{\mathcal{A}}(q, r) \cap K$ and $v \in L_{\mathcal{A}}(s, t) \cap K$. We show that $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}(R)$. By definition of \mathbf{K} , we have $u, v \in L$ and $u \sim v$. In particular, $u \in L_{\mathcal{A}}(q, r) \cap L$ which yields $u \in H_{q,r}$ by definition in Lemma 28. Together with $u \sim v$, this yields $v \in H_{q,r}$. Hence, $L_{\mathcal{A}}(s, t) \cap H_{q,r} \neq \emptyset$ and Lemma 28 yields $L_{\mathcal{B}_R}((q, s, q), (r, t, r)) \cap L \neq \emptyset$. One may now use a symmetrical argument to obtain $L_{\mathcal{B}_R}((s, q, s), (t, r, t)) \cap L \neq \emptyset$. By definition of L in Lemma 27, this yields $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}(R)$, completing the proof.

5 Conclusion

In this paper, we proved that for every group prevariety \mathcal{G} , there exist generic polynomial time Turing reductions from $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -separation to \mathcal{G} -separation, for input languages represented by NFAs. While a generic reduction from $BPol(\mathcal{G})$ -separation to \mathcal{G} -separation was already developed in [21], it relied on an involved machinery, which required to dig into a more general problem than $BPol(\mathcal{G})$ -separation, namely “ $BPol(\mathcal{G})$ -covering”. In particular, the techniques from [21] do not provide any way to build separators in $BPol(\mathcal{G})$ (when they exist). They also yield poor upper complexity bounds. At last, the results of [21] do not apply to $BPol(\mathcal{G}^+)$. In this case, even the existence of a generic reduction is new. It would be interesting to unify ideas of the present paper with the techniques of [21], to lift them to the setting of $BPol(\mathcal{G})$ - and $BPol(\mathcal{G}^+)$ -covering. We leave this for further work.

Our results imply that separation is decidable in *polynomial time* for a number of standard classes: the piecewise testable languages (*i.e.*, $BPol(\text{ST})$ or equivalently $\mathcal{B}\Sigma_1(<)$), the languages of dot-depth one (*i.e.*, $BPol(\text{ST}^+)$ or equivalently $\mathcal{B}\Sigma_1(<, +1)$), the classes $BPol(\text{MOD})$ and $BPol(\text{MOD}^+)$ (*i.e.*, $\mathcal{B}\Sigma_1(<, \text{MOD})$ and $\mathcal{B}\Sigma_1(<, +1, \text{MOD})$) and the classes $BPol(\text{GR})$ and $BPol(\text{GR}^+)$. While this was well-known for the piecewise testable languages [17, 6], all other results are new – not only regarding the complexity, but even regarding the decidability. Actually, it is shown in [12] that $BPol(\text{ST})$ -separation is P-complete. It turns out that the reduction of [12], from the circuit value problem, adapts to prove the P-completeness of separation for all of the above classes (we leave the details for further work). Finally, our results also apply to the classes $BPol(\text{AMT})$ and $BPol(\text{AMT}^+)$ (*i.e.*, $\mathcal{B}\Sigma_1(<, \text{AMOD})$ and $\mathcal{B}\Sigma_1(<, +1, \text{AMOD})$): we obtain that separation is in co-NP. While this is currently unknown, we conjecture that this is a *tight* upper bound. Indeed, it is known that AMT-separation is co-NP-complete [26].

References

- 1 Jorge Almeida and Marc Zeitoun. The pseudovariety \mathbf{J} is hyperdecidable. *RAIRO Theoretical Informatics and Applications*, 31(5):457–482, 1997.
- 2 Mustapha Arfi. Polynomial operations on rational languages. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*, STACS’87, pages 198–206, Berlin, Heidelberg, 1987. Springer-Verlag.
- 3 Janusz A. Brzozowski and Rina S. Cohen. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.
- 4 Antonio Cano, Giovanna Guaiana, and Jean-Eric Pin. Regular languages and partial commutations. *Journal of Information and Computation*, 230:76–96, 2013.
- 5 Laura Chaubard, Jean Éric Pin, and Howard Straubing. First order formulas with modular predicates. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS’06)*, pages 211–220, 2006.
- 6 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming*, ICALP’13, pages 150–161, Berlin, Heidelberg, 2013. Springer-Verlag.
- 7 Samuel Eilenberg. *Automata, Languages, and Machines*, volume B. Academic Press, Inc., Orlando, FL, USA, 1976.
- 8 Karsten Henckell, Stuart Margolis, Jean-Eric Pin, and John Rhodes. Ash’s type II theorem, profinite topology and Malcev products. *International Journal of Algebra and Computation*, 1:411–436, 1991.
- 9 Robert Knast. A semigroup characterization of dot-depth one languages. *RAIRO - Theoretical Informatics and Applications*, 17(4):321–330, 1983.
- 10 Alexis Maciel, Pierre Péladéau, and Denis Thérien. Programs over semigroups of dot-depth one. *Theoretical Computer Science*, 245(1):135–148, 2000.
- 11 Stuart Margolis and Jean-Eric Pin. Product of Group Languages. In *FCT’85*, volume 199 of *Lect. Notes Comp. Sci.*, pages 285–299. Springer-Verlag, 1985.
- 12 Tomáš Masopust. Separability by piecewise testable languages is ptime-complete. *Theor. Comput. Sci.*, 711:109–114, 2018.
- 13 Jean-Eric Pin. Algebraic tools for the concatenation product. *Theoretical Computer Science*, 292:317–342, 2003.
- 14 Jean-Eric Pin. An explicit formula for the intersection of two polynomials of regular languages. In *DLT 2013*, volume 7907 of *Lect. Notes Comp. Sci.*, pages 31–45. Springer, 2013.
- 15 Jean-Eric Pin and Howard Straubing. Some results on \mathcal{C} -varieties. *RAIRO - Theoretical Informatics and Applications*, 39(1):239–262, 2005.
- 16 Thomas Place, Varun Ramanathan, and Pascal Weil. Covering and separation for logical fragments with modular predicates. *Logical Methods in Computer Science*, 15(2), 2019.
- 17 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science*, MFCS’13, pages 729–740, Berlin, Heidelberg, 2013. Springer-Verlag.
- 18 Thomas Place and Marc Zeitoun. Separation for dot-depth two. In *Proceedings of the 32th Annual ACM/IEEE Symposium on Logic in Computer Science*, (LICS’17), pages 202–213. IEEE Computer Society, 2017.
- 19 Thomas Place and Marc Zeitoun. The covering problem. *Logical Methods in Computer Science*, 14(3), 2018.
- 20 Thomas Place and Marc Zeitoun. Generic results for concatenation hierarchies. *Theory of Computing Systems (ToCS)*, 63(4):849–901, 2019. Selected papers from CSR’17.
- 21 Thomas Place and Marc Zeitoun. Separation and covering for group based concatenation hierarchies. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS’19, pages 1–13, 2019.

- 22 Thomas Place and Marc Zeitoun. Adding successor: A transfer theorem for separation and covering. *ACM Transactions on Computational Logic*, 21(2):9:1–9:45, 2020.
- 23 Thomas Place and Marc Zeitoun. Separation for dot-depth two. *Logical Methods in Computer Science*, Volume 17, Issue 3, 2021.
- 24 Thomas Place and Marc Zeitoun. Characterizing level one in group-based concatenation hierarchies. In *Computer Science – Theory and Applications*, Cham, 2022. Springer International Publishing.
- 25 Thomas Place and Marc Zeitoun. A generic polynomial time approach to separation by first-order logic without quantifier alternation, 2022. doi:10.48550/arXiv.2210.00946.
- 26 Thomas Place and Marc Zeitoun. Group separation strikes back. To appear, a preliminary version is available at <https://www.labri.fr/perso/tplace/Files/groups.pdf>, 2022.
- 27 Imre Simon. Piecewise testable events. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pages 214–222, Berlin, Heidelberg, 1975. Springer-Verlag.
- 28 Benjamin Steinberg. Inevitable graphs and profinite topologies: Some solutions to algorithmic problems in monoid and automata theory, stemming from group theory. *International Journal of Algebra and Computation*, 11(1):25–72, 2001.
- 29 Howard Straubing. A generalization of the schützenberger product of finite monoids. *Theoretical Computer Science*, 13(2):137–150, 1981.
- 30 Howard Straubing. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.
- 31 Howard Straubing. On logical descriptions of regular languages. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics, LATIN’02*, pages 528–538, Berlin, Heidelberg, 2002. Springer-Verlag.
- 32 Denis Thérien. Classification of finite monoids: The language approach. *Theoretical Computer Science*, 14(2):195–208, 1981.
- 33 Gabriel Thierrin. Permutation automata. *Theory of Computing Systems*, 2(1):83–90, 1968.
- 34 Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.
- 35 Bret Tilson. Categories as algebra: essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48(1):83–198, 1987.
- 36 Georg Zetsche. Separability by piecewise testable languages and downward closures beyond subwords. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS’18*, pages 929–938, 2018.

A Appendix

In this appendix, we present the proof of Theorem 22. Let us first recall the statement.

► **Theorem 22.** *Let \mathcal{G} be a group prevariety and $\mathcal{A} = (Q, \delta)$ an NFA. Then, $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$ is the greatest $(BPol, +)$ -sound subset of Q^4 for \mathcal{G} and \mathcal{A} .*

The proof argument is based on the same outline as the one presented for Theorem 20 in the main paper. We fix a group prevariety \mathcal{G} and an NFA $\mathcal{A} = (Q, \delta)$. Let $S \subseteq Q^4$ be the greatest $(BPol, +)$ -sound subset for \mathcal{G} and \mathcal{A} . We prove that $S = \mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$.

First part: $S \subseteq \mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$. We use *tuple separation* and Lemma 17. Let us start with terminology. For every $n \geq 1$ and $(q_1, r_1, q_2, r_2) \in Q^4$, we associate an n -tuple $T_n(q_1, r_1, q_2, r_2)$. We use induction on n and tuple concatenation to present the definition. If $n = 1$ then, $T_1(q_1, r_1, q_2, r_2) = (L_{\mathcal{A}}(q_2, r_2))$. If $n > 1$, then,

$$T_n(q_1, r_1, q_2, r_2) = \begin{cases} (L_{\mathcal{A}}(q_2, r_2)) \cdot T_{n-1}(q_1, r_1, q_2, r_2) & \text{if } n \text{ is odd} \\ (L_{\mathcal{A}}(q_1, r_1)) \cdot T_{n-1}(q_1, r_1, q_2, r_2) & \text{if } n \text{ is even.} \end{cases}$$

We use induction on n to prove the following proposition.

► **Proposition 29.** *For every $n \geq 1$ and $(q_1, r_1, q_2, r_2) \in S$, the n -tuple $T_n(q_1, r_1, q_2, r_2)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable under \mathcal{G} -control.*

By definition, Proposition 29 implies that for every $p \geq 1$ and every $(q_1, r_1, q_2, r_2) \in S$, the $2p$ -tuple $(L_{\mathcal{A}}(q_1, r_1), L_{\mathcal{A}}(q_2, r_2))^p$ is not $\text{Pol}(\mathcal{G}^+)$ -separable under \mathcal{G} -control. By Corollary 15, it follows that $L_{\mathcal{A}}(q_1, r_1)$ is not $B\text{Pol}(\mathcal{G}^+)$ -separable from $L_{\mathcal{A}}(q_2, r_2)$ under \mathcal{G} -control, *i.e.* that $(q_1, r_1, q_2, r_2) \in \mathcal{I}_{B\text{Pol}(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$. We get $S \subseteq \mathcal{I}_{B\text{Pol}(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}]$ as desired.

We prove Proposition 29 using induction on n . We fix $n \geq 1$ for the proof. In order to exploit the fact that S is $(B\text{Pol}, +)$ -sound, we need a property of the NFA $\mathcal{B}_S^+ = (Q^3, \gamma_S)$ used to define $\tau_{\mathcal{A}, \mathcal{G}}^+$. When $n \geq 2$, this is where we use induction on n and Lemma 17.

► **Lemma 30.** *Consider $(s_1, s_2, s_3), (t_1, t_2, t_3) \in Q^3$ and a group language $H \subseteq A^*$. Assume that $H \cap L_{\mathcal{B}_S^+}((s_1, s_2, s_3), (t_1, t_2, t_3)) \neq \emptyset$. Then, $H \cap L_{\mathcal{A}}(s_1, t_1) \neq \emptyset$ and, if $n \geq 2$, then the n -tuple $(H \cap L_{\mathcal{A}}(s_1, t_1)) \cdot T_{n-1}(s_2, t_2, s_3, t_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable.*

Proof. By hypothesis, there exists $w \in H \cap L_{\mathcal{B}_S^+}((s_1, s_2, s_3), (t_1, t_2, t_3))$. Hence, the NFA \mathcal{B}_S^+ contains some run labeled by w from (s_1, s_2, s_3) to (t_1, t_2, t_3) . We use a sub-induction on the number of transitions involved in that run. When no transitions are used: we have $w = \varepsilon$ and $(s_1, s_2, s_3) = (t_1, t_2, t_3)$. It follows that $w = \varepsilon \in H \cap L_{\mathcal{A}}(s_1, t_1)$. Moreover, if $n \geq 2$, the n -tuple $(H \cap L_{\mathcal{A}}(s_1, t_1)) \cdot T_{n-1}(s_2, s_2, s_3, s_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable by Lemma 13 since $\varepsilon \in L_{\mathcal{A}}(s_2, s_2) \cap L_{\mathcal{A}}(s_3, s_3)$. We now assume that at least one transition is used. We get a triple $(q_1, q_2, q_3) \in Q^3$, a word $w' \in A^*$ and $x \in A \cup \{\varepsilon\}$ such that we have $w = w'x$, $w' \in L_{\mathcal{B}_S^+}((s_1, s_2, s_3), (q_1, q_2, q_3))$ and $((q_1, q_2, q_3), x, (t_1, t_2, t_3)) \in \gamma_S^+$. Since H is a group language, it is recognized by a morphism $\alpha : A^* \rightarrow G$ into a finite group G . Let $H' = \alpha^{-1}(\alpha(w'))$. Clearly, H' is a group language and $w' \in H' \cap L_{\mathcal{B}_S^+}((s_1, s_2, s_3), (q_1, q_2, q_3))$. Thus, induction yields that $H' \cap L_{\mathcal{A}}(s_1, q_1) \neq \emptyset$ and, if $n \geq 2$, the n -tuple $(H' \cap L_{\mathcal{A}}(s_1, q_1)) \cdot T_{n-1}(s_2, q_2, s_3, q_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable. We now consider two cases depending on $x \in A \cup \{\varepsilon\}$.

Assume first that $x = a \in A$: we have $((q_1, q_2, q_3), a, (t_1, t_2, t_3)) \in \gamma_S^+$. By definition, it follows that $(q_i, a, t_i) \in \delta$ for $i = \{1, 2, 3\}$. Observe that $(H' \cap L_{\mathcal{A}}(s_1, q_1))a \subseteq H \cap L_{\mathcal{A}}(s_1, t_1)$. Indeed, if $u \in (H' \cap L_{\mathcal{A}}(s_1, q_1))a$, then $u = u'a$ where $u' \in H'$ and $u' \in L_{\mathcal{A}}(s_1, q_1)$. Since $H' = \alpha^{-1}(\alpha(w'))$, the hypothesis that $u' \in H'$ yields $\alpha(u) = \alpha(u'a) = \alpha(w'a) = \alpha(w)$ which implies that $u \in H$ since $w \in H$ and H is recognized by α . Moreover, since $u' \in L_{\mathcal{A}}(s_1, q_1)$ and $(q_1, a, t_1) \in \delta$, we get $u = u'a \in L_{\mathcal{A}}(s_1, t_1)$. Altogether, this yields $u \in H \cap L_{\mathcal{A}}(s_1, t_1)$ as desired. Since we already know that $H' \cap L_{\mathcal{A}}(s_1, q_1) \neq \emptyset$, we get $H \cap L_{\mathcal{A}}(s_1, t_1) \neq \emptyset$. Moreover, if $n \geq 2$, since $(q_2, a, t_2), (q_3, a, t_3) \in \delta$, Lemma 13 yields that $(\{a\}) \cdot T_{n-1}(q_2, t_2, q_3, t_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable. Hence, since we already know that $(H' \cap L_{\mathcal{A}}(s_1, q_1)) \cdot T_{n-1}(s_2, q_2, s_3, q_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable and $(H' \cap L_{\mathcal{A}}(s_1, q_1))a \subseteq H \cap L_{\mathcal{A}}(s_1, t_1)$, it follows from Lemma 14 that $(H \cap L_{\mathcal{A}}(s_1, t_1)) \cdot T_{n-1}(s_2, t_2, s_3, t_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable.

Finally, assume that $x = \varepsilon$: we have $((q_1, q_2, q_3), \varepsilon, (t_1, t_2, t_3)) \in \gamma_S^+$. By definition, it follows that $q_1 = t_1$, $(q_2, t_2, q_3, t_3) \in S$ and there exists a nonempty word $y \in A^+$ which belongs to $L_{\mathcal{A}}(q_1, q_1)$, $L_{\mathcal{A}}(q_2, q_2)$, $L_{\mathcal{A}}(q_3, q_3)$, $L_{\mathcal{A}}(t_2, t_2)$ and $L_{\mathcal{A}}(t_3, t_3)$. Since $x = \varepsilon$, we have $w = w'$. Hence, since $w \in H$ and H is recognized by α , we obtain that $H' = \alpha(\alpha^{-1}(w')) \subseteq H$. Since $H' \cap L_{\mathcal{A}}(s_1, q_1) \neq \emptyset$ and $q_1 = t_1$, we get $H \cap L_{\mathcal{A}}(s_1, t_1) \neq \emptyset$. We now assume that $n \geq 2$. Since G is a finite group, there exists $k \geq 1$ such that $\alpha(y^k) = 1_G$. We write $z = y^k$. By hypothesis on y , we also have $z \in L_{\mathcal{A}}(q_1, q_1)$. It follows that $z^+ \subseteq \alpha^{-1}(1_G) \cap L_{\mathcal{A}}(q_1, q_1)$. Additionally, since z belongs to $L_{\mathcal{A}}(q_2, q_2)$, $L_{\mathcal{A}}(q_3, q_3)$, $L_{\mathcal{A}}(t_2, t_2)$ and $L_{\mathcal{A}}(t_3, t_3)$, we know that $z^+ L_{\mathcal{A}}(q_2, t_2) z^+ \subseteq L_{\mathcal{A}}(q_2, t_2)$ and $z^+ L_{\mathcal{A}}(q_3, t_3) z^+ \subseteq L_{\mathcal{A}}(q_3, t_3)$. Since $(q_2, t_2, q_3, t_3) \in S$, it follows from induction on n in Proposition 29 that the $(n-1)$ -tuple $T_{n-1}(q_2, t_2, q_3, t_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable under \mathcal{G} -control. Altogether, we obtain from Lemma 17 that the n -tuple $(\alpha^{-1}(1_G) \cap L_{\mathcal{A}}(q_1, q_1)) \cdot T_{n-1}(q_2, t_2, q_3, t_3)$ is not $\text{Pol}(\mathcal{G}^+)$ -separable. Finally, since $q_1 = t_1$

and $H' \subseteq H$, one may verify that $(H' \cap L_{\mathcal{A}}(s_1, q_1))(\alpha^{-1}(1_G) \cap L_{\mathcal{A}}(q_1, q_1)) \subseteq (H \cap L_{\mathcal{A}}(s_1, t_1))$. Since we already know that $(H' \cap L_{\mathcal{A}}(s_1, q_1)) \cdot T_{n-1}(s_2, q_2, s_3, q_3)$ is not $Pol(\mathcal{G}^+)$ -separable, Lemma 14 yields that $(H \cap L_{\mathcal{A}}(s_1, t_1)) \cdot T_{n-1}(s_2, t_2, s_3, t_3)$ is not $Pol(\mathcal{G}^+)$ -separable. \blacktriangleleft

We may now complete the proof of Proposition 29. By symmetry, we only treat the case when n is odd and leave the even case to the reader. Let $(q_1, r_1, q_2, r_2) \in S$, we have to prove that $T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G}^+)$ -separable under \mathcal{G} -control. Hence, we fix $H \in \mathcal{G}$ such that $\varepsilon \in H$ and prove $H \cap T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G}^+)$ -separable. Since S is $(BPol, +)$ -sound, we have $\tau_{\mathcal{A}, \mathcal{G}}^+(S) = S$ which implies that $(q_1, r_1, q_2, r_2) \in \tau_{\mathcal{A}, \mathcal{G}}^+(S)$. Hence, it follows from (2) that $\{\varepsilon\}$ is not \mathcal{G} -separable from $L_{\mathcal{B}_S^+}((q_2, q_1, q_2), (r_2, r_1, r_2))$. Since $H \in \mathcal{G}$ and $\varepsilon \in H$, it follows that $H \cap L_{\mathcal{B}_S^+}((q_2, q_1, q_2), (r_2, r_1, r_2)) \neq \emptyset$. If $n = 1$, Lemma 30 yields $H \cap L_{\mathcal{A}}(q_2, r_2) \neq \emptyset$. Since $T_1(q_1, r_1, q_2, r_2) = (L_{\mathcal{A}}(q_2, r_2))$, we get that $H \cap T_1(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G}^+)$ -separable as desired. If $n \geq 2$, then Lemma 30 implies that $(H \cap L_{\mathcal{A}}(s_1, t_1)) \cdot T_{n-1}(s_2, t_2, s_3, t_3)$ is not $Pol(\mathcal{G}^+)$ -separable. Thus, since $H \in \mathcal{G} \subseteq Pol(\mathcal{G}^+)$, one may verify that the n -tuple $(H \cap L_{\mathcal{A}}(q_2, r_2)) \cdot (H \cap T_{n-1}(q_1, r_1, q_2, r_2))$ is not $Pol(\mathcal{G}^+)$ -separable. By definition, this exactly says that $H \cap T_n(q_1, r_1, q_2, r_2)$ is not $Pol(\mathcal{G}^+)$ -separable, completing the proof.

Second part: $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}] \subseteq S$. Consider an arbitrary set $R \subseteq Q^4$. We say that R is multiplication-closed to indicate that for every $(q, r, s, t) \in R$ and $(q', r', s', t') \in R$, if $r = q'$ and $t = s'$, then $(q, r', s, t') \in R$. Moreover, we say that an arbitrary set $R \subseteq Q^4$ is *good* if it is multiplication-closed and there are $L \in \mathcal{G}$ such $\varepsilon \in L$ and a $BPol(\mathcal{G}^+)$ -cover \mathbf{K} of L which is separating for R .

► **Proposition 31.** *Let $R \subseteq Q^4$. If R is good, then $\tau_{\mathcal{A}, \mathcal{G}}^+(R)$ is good as well.*

We use Proposition 31 to complete the proof. Let $S_0 = Q^4$ and $S_i = \tau_{\mathcal{A}, \mathcal{G}}^+(S_{i-1})$ for $i \geq 1$. By Lemma 21, we have $S_0 \supseteq S_1 \subseteq S_2 \supseteq \dots$ and there is $n \in \mathbb{N}$ such that S_n is the greatest $(BPol, +)$ -sound subset for \mathcal{G} and \mathcal{A} , i.e. such that $S_n = S$. Since S_0 is good (it is clearly multiplication-closed and $\{A^*\}$ is a $BPol(\mathcal{G}^+)$ -cover of $A^* \in \mathcal{G}$ which is separating for $S_0 = Q^4$), Proposition 31 implies that S_i is good for all $i \in \mathbb{N}$. Hence, $S = S_n$ is good. We get $L \in \mathcal{G}$ such $\varepsilon \in L$ and a $BPol(\mathcal{G}^+)$ -cover \mathbf{K} of L which is separating for S . By Lemma 11, this yields $\mathcal{I}_{BPol(\mathcal{G}^+)}[\mathcal{G}, \mathcal{A}] \subseteq S$ as desired.

We turn to Proposition 25. Let $R \subseteq Q^4$ be a good set. We have to prove that $\tau_{\mathcal{A}, \mathcal{G}}^+(R)$ is multiplication-closed and build $L \in \mathcal{G}$ such $\varepsilon \in L$ and a $BPol(\mathcal{G}^+)$ -cover \mathbf{K} of L which is separating for $\tau_{\mathcal{A}, \mathcal{G}}^+(R)$. This proves that $\tau_{\mathcal{A}, \mathcal{G}}^+(R)$ is good as desired. Let us first prove that $\tau_{\mathcal{A}, \mathcal{G}}^+(R)$ is multiplication-closed (we use the hypothesis that R is good).

► **Lemma 32.** *The set $\tau_{\mathcal{A}, \mathcal{G}}^+(R) \subseteq Q^4$ is multiplication-closed.*

Proof. Let $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}^+(R)$ and $(q', r', s', t') \in \tau_{\mathcal{A}, \mathcal{G}}^+(R)$ such that $r = q'$ and $t = s'$. We need to prove that $(q, r', s, t') \in \tau_{\mathcal{A}, \mathcal{G}}^+(R)$. By (2) in the definition, this boils down to proving that $\{\varepsilon\}$ is not \mathcal{G} -separable from $L_{\mathcal{B}_R^+}((s, q, s), (t', r', t'))$ and $L_{\mathcal{B}_R^+}((q, s, q), (r', t', r'))$. By symmetry, we only prove the former. By hypothesis on (q, r, s, t) and (q', r', s', t') , we get from (2) that $\{\varepsilon\}$ is not \mathcal{G} -separable from both $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t))$ and $L_{\mathcal{B}_R^+}((s', q', s'), (t', r', t'))$. Since \mathcal{G} is a prevariety it then follows from Lemma 14 that $\{\varepsilon\}$ is not \mathcal{G} -separable from the concatenation $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t))L_{\mathcal{B}_R^+}((s', q', s'), (t', r', t'))$. Finally, since $(t, r, t) = (s', q', s')$, we know that $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t))L_{\mathcal{B}_R^+}((s', q', s'), (t', r', t')) \subseteq L_{\mathcal{B}_R^+}((s, q, s), (t', r', t'))$. We conclude that $\{\varepsilon\}$ is not \mathcal{G} -separable from both $L_{\mathcal{B}_R^+}((s, q, s), (t', r', t'))$ as desired. \blacktriangleleft

We now build $L \in \mathcal{G}$ such that $\varepsilon \in L$ (this part is independent from our hypothesis on R).

► **Lemma 33.** *There exists $L \in \mathcal{G}$ such that $\varepsilon \in L$ and for every $(q, r, s, t) \in Q^4$, if $L_{\mathcal{B}_R^+}((q, s, q), (r, t, r)) \cap L \neq \emptyset$ and $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t)) \cap L \neq \emptyset$, then $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}^+(R)$.*

Proof. Let \mathbf{H} be the finite set of all languages recognized by \mathcal{B}_R^+ such that $\{\varepsilon\}$ is \mathcal{G} -separable from H . For every $H \in \mathbf{H}$, there exists $L_H \in \mathcal{G}$ such that $\varepsilon \in L_H$ and $L_H \cap H = \emptyset$. We define $L = \bigcap_{H \in \mathbf{H}} L_H \in \mathcal{G}$. It is clear that $\varepsilon \in L$. Moreover, given $(q, r, s, t) \in Q^4$, if $L_{\mathcal{B}_R^+}((q, s, q), (r, t, r)) \cap L \neq \emptyset$ and $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t)) \cap L \neq \emptyset$, it follows from the definition of L that $\{\varepsilon\}$ is not \mathcal{G} -separable from both $L_{\mathcal{B}_R^+}((q, s, q), (r, t, r))$ and $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t))$. It then follows from (2) in the definition of $\tau_{\mathcal{A}, \mathcal{G}}^+$ that $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}^+(R)$. ◀

We fix $L \in \mathcal{G}$ as described in Lemma 33 for the remainder of the proof. We now build the $BPol(\mathcal{G}^+)$ -cover \mathbf{K} of L using the hypothesis that R is good and Proposition 7.

► **Lemma 34.** *For all $(q, r) \in Q^2$, there is $H_{q,r} \in BPol(\mathcal{G}^+)$ such that $L_{\mathcal{A}}(q, r) \cap L \subseteq H_{q,r}$ and for all pairs $(s, t) \in Q^2$, if $L_{\mathcal{A}}(s, t) \cap H_{q,r} \neq \emptyset$ then $L_{\mathcal{B}_R^+}((q, s, q), (r, t, r)) \cap L \neq \emptyset$.*

Proof. Since R is good, there are $U \in \mathcal{G}$ such that $\varepsilon \in U$ and a $BPol(\mathcal{G}^+)$ -cover \mathbf{V} of U which is separating for R . We use them to build $H_{q,r}$. Since $U \in \mathcal{G}$ and $\varepsilon \in U$ Proposition 7 yields a cover \mathbf{P} of $L_{\mathcal{A}}(q, r) \cap L$ such that for each $P \in \mathbf{P}$, there exists a word $w_P \in L_{\mathcal{A}}(q, r) \cap L$ and an \mathcal{A} -guarded decomposition (w_1, \dots, w_{n+1}) of w_P for some $n \in \mathbb{N}$ such that $P = w_1 U \dots w_n U w_{n+1}$ (if $n = 0$, then $P = \{w_1\}$). Now, for every $P \in \mathbf{P}$, we build a $BPol(\mathcal{G}^+)$ -cover \mathbf{K}_P of P from the cover \mathbf{V} of U . Let (w_1, \dots, w_{n+1}) be the \mathcal{A} -guarded decomposition of w_P such that $P = w_1 U \dots w_n U w_{n+1}$ (in particular, this means that P is of the form $U_0 a_1 U_1 \dots a_m U_m$ where $a_1 \dots a_m = w_1 \dots w_n$ and $U_i = U$ or $U_i = \{\varepsilon\}$ for each $i \leq m$). By definition, \mathbf{V} is a $BPol(\mathcal{G}^+)$ -cover of $U \in \mathcal{G} \subseteq Pol(\mathcal{G}^+)$. Moreover, we have $\{\varepsilon\} \in \mathcal{G}^+ \subseteq Pol(\mathcal{G}^+)$ by definition of \mathcal{G}^+ and $\{\{\varepsilon\}\}$ is a $BPol(\mathcal{G}^+)$ -cover of $\{\varepsilon\}$. Hence, Proposition 5 yields a $BPol(\mathcal{G}^+)$ -cover \mathbf{K}_P of $P = w_1 U \dots w_n U w_{n+1}$ such that for every $K \in \mathbf{K}_P$, there exist $V_1, \dots, V_n \in \mathbf{V}$ such that $K \subseteq w_1 V_1 \dots w_n V_n w_{n+1}$. We define $H_{q,r}$ as the union of all languages K such that $K \in \mathbf{K}_P$ for some $P \in \mathbf{P}$ and $L_{\mathcal{A}}(q, r) \cap K \neq \emptyset$. Clearly, $H_{q,r} \in BPol(\mathcal{G}^+)$. Moreover, since \mathbf{P} is a cover of $L_{\mathcal{A}}(q, r) \cap L$, and \mathbf{K}_P is a cover of P for each $P \in \mathbf{P}$, it is clear that $L_{\mathcal{A}}(q, r) \cap L \subseteq H_{q,r}$. We now fix $(s, t) \in Q^2$ such that $L_{\mathcal{A}}(s, t) \cap H_{q,r} \neq \emptyset$ and show that $L_{\mathcal{B}_R^+}((q, s, q), (r, t, r)) \cap L \neq \emptyset$. By definition of $H_{q,r}$, we get $P \in \mathbf{P}$ and $K \in \mathbf{K}_P$ such that $L_{\mathcal{A}}(q, r) \cap K \neq \emptyset$ and $L_{\mathcal{A}}(s, t) \cap K \neq \emptyset$. By definition, $P = w_1 U \dots w_n U w_{n+1}$ where (w_1, \dots, w_{n+1}) is an \mathcal{A} -guarded decomposition of $w_P \in L_{\mathcal{A}}(q, r) \cap L$. We use w_P to build a new word $w' \in L_{\mathcal{B}_R^+}((q, s, q), (r, t, r)) \cap L$.

We fix $x \in L_{\mathcal{A}}(s, t) \cap K$ and $y \in L_{\mathcal{A}}(q, r) \cap K$. Since $w_P = w_1 \dots w_{n+1}$ and $w_P \in L_{\mathcal{A}}(q, r)$, we may decompose the corresponding run in \mathcal{A} : we get $p_0, \dots, p_{n+1} \in Q$ such that $p_0 = q$, $p_{n+1} = r$ and $w_i \in L_{\mathcal{A}}(p_{i-1}, p_i)$ for $1 \leq i \leq n+1$. Moreover, since $K \in \mathbf{K}_P$, we have $K \subseteq w_1 V_1 \dots w_n V_n w_{n+1}$ for $V_1, \dots, V_n \in \mathbf{V}$ (if $n = 0$, then $K \subseteq \{w_1\}$). Since $x, y \in K$, we get $x_i, y_i \in V_i$ for $1 \leq i \leq n$ such that $x = w_1 x_1 \dots w_n x_n w_{n+1}$ and $y = w_1 y_1 \dots w_n y_n w_{n+1}$. Since $x \in L_{\mathcal{A}}(s, t)$, we get $s_1, t_1, \dots, s_{n+1}, t_{n+1} \in Q$ where $s_1 = s$, $t_{n+1} = t$, $w_i \in L_{\mathcal{A}}(s_i, t_i)$ for $1 \leq i \leq n+1$ and $x_i \in L_{\mathcal{A}}(t_i, s_{i+1})$ for $1 \leq i \leq n$. Symmetrically, since $y \in L_{\mathcal{A}}(q, r)$, we get $q_1, r_1, \dots, q_{n+1}, r_{n+1} \in Q$ with $q_1 = q$, $r_{n+1} = r$, $w_i \in L_{\mathcal{A}}(q_i, r_i)$ for $1 \leq i \leq n+1$, and $y_i \in L_{\mathcal{A}}(r_i, q_{i+1})$ for $1 \leq i \leq n$. First, note that when $n = 0$, we have $w_P = w_1$ and the above implies that $w_P \in L_{\mathcal{A}}(q, r)$ and $w_P \in L_{\mathcal{A}}(s, t)$. Thus, $w_P \in L_{\mathcal{B}_R^+}((q, s, q), (r, t, r))$ by definition of the labeled transition in \mathcal{B}_R^+ . This concludes the proof since we also know that $w_P \in L$. We now assume that $n \geq 1$.

By hypothesis, (w_1, \dots, w_{n+1}) is an \mathcal{A} -guarded decomposition. Hence, for $1 \leq i \leq n$, we get $z_i \in A^+$ which is a right \mathcal{A} -loop for w_i and a left \mathcal{A} -loop for w_{i+1} . Let $\alpha : A^* \rightarrow G$ be a morphism into a finite group G recognizing both L and U (recall that L and U are group languages). Since g is a finite group, there exists $k \geq 1$ such that for each $1 \leq i \leq n$, we have $\alpha(z_i^k) = 1_G$. We let $u_i = z_i^k$ for $1 \leq i \leq n$. One may verify that u_i remains a right \mathcal{A} -loop for w_i and a left \mathcal{A} -loop for w_{i+1} . Moreover, since $\alpha(u_i) = 1_G$, we know that $u_i \in U$ (recall that $\varepsilon \in U$ and U is recognized by α). We let $w'_1 = w_1 u_1$, $w'_{n+1} = u_n w_{n+1}$ and $w'_i = u_{i-1} w_i u_i$ for $2 \leq i \leq n$. Finally, we let $w' = w'_1 \cdots w'_n w'_{n+1}$ and show that $w' \in L \cap L_{\mathcal{B}_R^+}((q, s, q), (r, t, r))$ which completes the proof. First, since $\alpha(u_i) = 1_G$ for $1 \leq i \leq n$, it is immediate that $\alpha(w') = \alpha(w_1 \cdots w_n w_{n+1}) = \alpha(w_P)$. Since $w_P \in L$ which is recognized by α , we get $w' \in L$.

We now concentrate on proving that $w' \in L_{\mathcal{B}_R^+}((q, s, q), (r, t, r))$. For $1 \leq i \leq n+1$, we know that w_i belongs to $L_{\mathcal{A}}(p_{i-1}, p_i)$, $L_{\mathcal{A}}(s_i, t_i)$ and $L_{\mathcal{A}}(q_i, r_i)$. Hence, one may verify from the definition of left/right \mathcal{A} -loops that there are $p'_0, \dots, p'_{n+1} \in Q$, $s'_1, t'_1, \dots, s'_{n+1}, t'_{n+1} \in Q$ and $q'_1, r'_1, \dots, q'_{n+1}, r'_{n+1} \in Q$ such that,

- $p'_0 = p_0 = q$, $p'_{n+1} = p_{n+1} = r$, $w'_i \in L_{\mathcal{A}}(p'_{i-1}, p'_i)$ for $1 \leq i \leq n+1$ and $u_i \in L_{\mathcal{A}}(p'_i, p'_i)$ for $1 \leq i \leq n$.
- $s'_0 = s_0 = s$, $t'_{n+1} = t_{n+1} = t$, $w'_i \in L_{\mathcal{A}}(s'_i, t'_i)$ for $1 \leq i \leq n+1$ and we have $u_i \in L_{\mathcal{A}}(t'_i, t'_i) \cap L_{\mathcal{A}}(t'_i, t_i) \cap L_{\mathcal{A}}(s_{i+1}, s'_{i+1}) \cap L_{\mathcal{A}}(s'_{i+1}, s'_{i+1})$ for $1 \leq i \leq n$.
- $q'_0 = q_0 = q$, $r'_{n+1} = r_{n+1} = r$, $w'_i \in L_{\mathcal{A}}(q'_i, r'_i)$ for $1 \leq i \leq n+1$ and we have $u_i \in L_{\mathcal{A}}(r'_i, r'_i) \cap L_{\mathcal{A}}(r'_i, r_i) \cap L_{\mathcal{A}}(q_{i+1}, q'_{i+1}) \cap L_{\mathcal{A}}(q'_{i+1}, q'_{i+1})$ for $1 \leq i \leq n$.

By definition of the labeled transitions in the NFA \mathcal{B}_R^+ , it is straightforward to verify that we have $w'_i \in L_{\mathcal{B}_R^+}((p'_{i-1}, s'_i, q'_i), (p'_i, t'_i, r'_i))$ for $1 \leq i \leq n+1$. We now prove the following fact.

► **Fact 35.** For $1 \leq i \leq n$, we have $((p'_i, t'_i, r'_i), \varepsilon, (p'_i, s'_{i+1}, q'_{i+1})) \in \gamma_R^+$.

Proof. We fix i for the proof. Since we know that $u_i \in A^+$ belongs to $L_{\mathcal{A}}(p'_i, p'_i)$, $L_{\mathcal{A}}(t'_i, t'_i)$, $L_{\mathcal{A}}(r'_i, r'_i)$, $L_{\mathcal{A}}(s'_{i+1}, s'_{i+1})$ and $L_{\mathcal{A}}(q'_{i+1}, q'_{i+1})$, it suffices to prove that $(t'_i, s'_{i+1}, r'_i, q'_{i+1}) \in R$. This will imply that $((p'_i, t'_i, r'_i), \varepsilon, (p'_i, s'_{i+1}, q'_{i+1})) \in \gamma_R^+$ by definition of γ_R^+ . Recall that $x_i \in L_{\mathcal{A}}(t_i, s_{i+1})$, $y_i \in L_{\mathcal{A}}(r_i, q_{i+1})$ and $x_i, y_i \in V_i$. Since $V_i \in \mathbf{V}$ which is *separating* for R , it follows that $(t_i, s_{i+1}, r_i, q_{i+1}) \in R$. Moreover, $u_i \in U$ which yields $V \in \mathbf{V}$ such that $u_i \in V$ since \mathbf{V} is a cover of U . Hence, since $u_i \in L_{\mathcal{A}}(t'_i, t_i)$ and $u_i \in L_{\mathcal{A}}(r'_i, r_i)$. The hypothesis that \mathbf{V} is separating for R also yields $(t'_i, t_i, r'_i, r_i) \in R$. Symmetrically, one may use the hypotheses that $u_i \in L_{\mathcal{A}}(s_{i+1}, s'_{i+1})$ and $u_i \in L_{\mathcal{A}}(q_{i+1}, q'_{i+1})$ to verify that $(s_{i+1}, s'_{i+1}, q_{i+1}, q'_{i+1}) \in R$. Altogether, since R is multiplication-closed, we get $(t'_i, s'_{i+1}, r'_i, q'_{i+1}) \in R$ as desired. ◀

In view of Fact 35, we obtain $w' = w'_1 \cdots w'_n w'_{n+1} \in L_{\mathcal{B}_R^+}((p'_0, s'_1, q'_1), (p'_{n+1}, t'_{n+1}, r'_{n+1}))$. This exactly says that $w' \in L_{\mathcal{B}_R^+}((q, s, q), (r, t, r))$ which completes the proof. ◀

We may now build \mathbf{K} . Let $\mathbf{H} = \{H_{q,r} \mid (q, r) \in Q^2\}$. Consider the following equivalence \sim defined on L : given $u, v \in L$, we let $u \sim v$ if and only if $u \in H_{q,r} \Leftrightarrow v \in H_{q,r}$ for every $(q, r) \in Q^2$. We let \mathbf{K} as the partition of L into \sim -classes. Clearly, each $K \in \mathbf{K}$ is a Boolean combination involving the languages in \mathbf{H} (which belong to $BPol(\mathcal{G}^+)$) and $L \in \mathcal{G}$. Hence, \mathbf{K} is a $BPol(\mathcal{G}^+)$ -cover of L . It remains to prove that it is separating for $\tau_{\mathcal{A}, \mathcal{G}}^+(R)$. Let $q, r, s, t \in Q$ and $K \in \mathbf{K}$ such that there are $u \in L_{\mathcal{A}}(q, r) \cap K$ and $v \in L_{\mathcal{A}}(s, t) \cap K$. By definition of \mathbf{K} , we have $u, v \in L$ and $u \sim v$. In particular, we have $u \in L_{\mathcal{A}}(q, r) \cap L$ which yields $u \in H_{q,r}$ by definition in Lemma 34. Together with $u \sim v$, this yields $v \in H_{q,r}$. Hence, $L_{\mathcal{A}}(s, t) \cap H_{q,r} \neq \emptyset$ and Lemma 34 yields $L_{\mathcal{B}_R^+}((q, s, q), (r, t, r)) \cap L \neq \emptyset$. One may now use a symmetrical argument to obtain $L_{\mathcal{B}_R^+}((s, q, s), (t, r, t)) \cap L \neq \emptyset$. By definition of L in Lemma 33, this yields $(q, r, s, t) \in \tau_{\mathcal{A}, \mathcal{G}}(R)$, completing the proof.

A Technique to Speed up Symmetric Attractor-Based Algorithms for Parity Games

K. S. Thejaswini ✉

Department of Computer Science, University of Warwick, Coventry, UK

Pierre Ohlmann ✉

University of Warsaw, Poland

Marcin Jurdziński ✉ 

Department of Computer Science, University of Warwick, Coventry, UK

Abstract

The classic McNaughton-Zielonka algorithm for solving parity games has excellent performance in practice, but its worst-case asymptotic complexity is worse than that of the state-of-the-art algorithms. This work pinpoints the mechanism that is responsible for this relative underperformance and proposes a new technique that eliminates it. The culprit is the wasteful manner in which the results obtained from recursive calls are indiscriminately discarded by the algorithm whenever subgames on which the algorithm is run change. Our new technique is based on firstly enhancing the algorithm to compute attractor decompositions of subgames instead of just winning strategies on them, and then on making it carefully use attractor decompositions computed in prior recursive calls to reduce the size of subgames on which further recursive calls are made. We illustrate the new technique on the classic example of the recursive McNaughton-Zielonka algorithm, but it can be applied to other symmetric attractor-based algorithms that were inspired by it, such as the quasi-polynomial versions of the McNaughton-Zielonka algorithm based on universal trees.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Design and analysis of algorithms; Theory of computation → Logic and verification

Keywords and phrases Parity games, Attractor decomposition, Quasipolynomial Algorithms, Universal trees

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.44

Related Version *Full Version*: <https://arxiv.org/abs/2010.08288> [30]

Funding EPSRC grant EP/P020992/1 (Solving Parity Games in Theory and Practice).

Pierre Ohlmann: Project BOBR that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 948057).

Acknowledgements We thank Rémi Morvan for contributing to several simulating discussions. We are grateful to Aditya Prakash for proof reading the current version of the paper, and to Nathanaël Fijalkow and Olivier Serre for doing the same with an earlier version of the paper. We also thank anonymous referees for pointing out some missing references from our previous draft as well as for their valuable comments to improve our current presentation. The authors are listed in reverse alphabetical order.

1 Context and contributions

Parity games are two-player games on graphs, which have been studied since early 1990’s [10, 11] and have many applications in automata theory on infinite trees [15], fixpoint logics [9, 4], verification and synthesis [28, 29]. They are intimately linked to the problems of emptiness and complementation of nondeterministic automata on trees [10, 33], model checking [11, 4, 16] and satisfiability checking of fixpoint logics, or fair simulation relations [12].



© K. S. Thejaswini, Pierre Ohlmann, and Marcin Jurdziński;

licensed under Creative Commons License CC-BY 4.0

42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

Editors: Anuj Dawar and Venkatesan Guruswami; Article No. 44; pp. 44:1–44:20



Leibniz International Proceedings in Informatics

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

Determining the winner of a parity games are one of the few problems known to lie in the complexity class $\text{NP} \cap \text{coNP}$ as well as $\text{UP} \cap \text{coUP}$ but not known to have a polynomial algorithm. Existence of a polynomial algorithm for solving parity games, which has been an important open problem for nearly three decades, has recently gained a lot of attention after the major breakthrough of Calude, Jain, Khoussainov, Li and Stephan [5], who provided a quasi-polynomial solution to the problem. Several algorithms [18, 23, 27, 24] followed this work, each providing a different perspective to solving parity games. Czerwiński, Daviaud, Fijalkow, Jurdziński, Lazić, and Parys [6] originally exhibited an underlying combinatorial structure of universal trees, provably underlying the techniques of Calude et al. of Jurdziński and Lazić, and of Lehtinen. Czerwiński et al. have also established a quasi-polynomial lower bound for the size of smallest universal trees, providing evidence that the techniques developed in these papers may be insufficient for leading to further improvements in the complexity of solving parity games. Following their work, Jurdziński, Morvan and Thejaswini [20] extended this result to formulate attractor based algorithms, as that of Parys [27] and also Lehtinen, Schewe, and Wojtczak [24] using universal trees too.

We propose a new technique for speeding up symmetric attractor-based algorithms for solving parity games. We focus on illustrating the technique on the example of the classic McNaughton-Zielonka algorithm [33], but we argue that it is applicable to other symmetric attractor-based algorithms that were inspired by McNaughton-Zielonka [1, 27, 25, 20, 22]. Many such algorithms have exhibited excellent performance in practice, significantly beating other classes of algorithms on standard benchmarks [21, 31]. On the other hand, their worst-case asymptotic running time is typically worse than that of asymmetric algorithms [17, 5, 18, 8]. More specifically, using the perspective of how various algorithms for parity games are related to universal trees [6], while the running time of state-of-the-art asymmetric algorithms is dominated by the size of a universal tree [18, 8], it is the square of the size of a universal tree for symmetric algorithms [20, 25]. Our technique allows to reduce the worst-case running time of symmetric attractor-based algorithms to match the linear dependence on the size of a universal tree enjoyed by asymmetric algorithms.

Our technique is based on making a better use of structural information obtained from earlier recursive calls to significantly reduce the worst-case overall size of the tree of recursive calls of the algorithm. While existing symmetric attractor-based algorithms are typically computing just the winning sets of positions or positional winning strategies, following [20], we propose to enhance them to explicitly record more finely structured witnesses of winning strategies called attractor decompositions. Moreover, we show how witnesses for both players from recursive calls on subgames can be meaningfully used to reduce the sizes of subgames on which further recursive calls are made, even if their key properties are damaged by the removal of some vertices from subgames on which they were computed. In contrast, other symmetric attractor-based algorithms are wasteful by routinely discarding witnesses for one of the players that are computed in recursive calls; in the worst case, this results in repeatedly solving large subgames from scratch. Our technique is robust and it applies to both the classic exponential-time McNaughton-Zielonka algorithm [33] and its more recent quasi-polynomial variants [27, 20, 25]. We are also confident that it is applicable to other symmetric attractor-based algorithms such as priority promotion [1]. Such algorithms can be interpreted as variants of the McNaughton-Zielonka algorithm that are enhanced by ad-hoc heuristics to construct attractor decompositions which are more robust to the wasteful behaviour described above. Our technique offers a more principled approach, in which decompositions of subgames computed in previous recursive calls are never discarded and are instead used in a systematic manner to speed up and reduce the number of further recursive calls.

2 Games, Strategies, Attractor decomposition

Parity Games. A parity game \mathcal{G} consists of a finite directed graph (V, E) , where the vertices are partitioned into V_{Even} and V_{Odd} where these belong to the Even player and the Odd player respectively along with a mapping π , from V to the set $\{1, \dots, h\}$ that labels every vertex with a positive integer, called its priority. A token is moved from a designated start vertex into a neighbour by Even or Odd depending on who owns the vertex, forming a sequence of vertices, which we will call a *play*. An infinite play is said to be winning for Even if the highest priority that occurs infinitely often is Even and Odd wins otherwise.

A (positional) *strategy* $\sigma \subseteq E$ for Even is a subset of edges originating from Even owned vertices. A game restricted to an Even strategy refers to the subgraph induced by considering only edges in σ along with edges in which originate in Odd's vertex. We call a cycle in a game \mathcal{G} an even cycle if the highest priority of the cycle is Even. A strategy is said to be winning for Even from a vertex v , if the game restricted to the Even strategy σ is such that the only cycles reachable from v are such that the highest priority in the cycle is even.

Attractors, traps, and dominions. In a parity game \mathcal{G} , for a target set of vertices B and a set of vertices A such that $B \subseteq A$, we say that an Even strategy σ is an Even reachability strategy to B from A if every infinite path in the subgraph restricted to σ along with all edges from all Odd vertices, that starts from a vertex in A contains at least one vertex in B . We call the largest such set A for which there is a reachability strategy, the *Even attractor* of B and we denote it by $\text{Attr}_{\text{Even}}^{\mathcal{G}}(B)$.

A trap for Odd is a subgame T of \mathcal{G} , where Even has a strategy σ , on restricted to which all paths from T remain in T . A subgame D in a game \mathcal{G} is said to be an *Even dominion* if it is an Odd trap and moreover, every cycle that can be reached from any vertex in D using the strategy used to trap Odd is an even cycle.

We also say that a set of vertices C is a *quasi-dominion* for Even if the subgame $\mathcal{G} \cap C$ is winning for Even. Note that all dominions of Even are also quasi-dominions but not all quasi-dominions of Even are dominions, since Odd might be able to escape a quasi-dominion.

If we do not mention if an object defined is for Even or for Odd, we assume Even by default. All of the above can be defined analogously for the other player.

Trees. Throughout this paper, trees only refer to rooted ordered trees. For a totally ordered set Σ , an *ordered tree* \mathcal{T} is a finite prefix closed set consisting of sequences of elements from Σ . We say that the *root* of the tree is the empty sequence $\langle \rangle$. We call an element of this prefix closed set, a *node* and use $\eta, \gamma, \epsilon \dots$ to refer to it. The leaves of a tree are the maximal elements of \mathcal{T} . For a node $\eta \in \mathcal{T}$, the subtree rooted at η is the tree $\eta^{-1} \cdot \mathcal{T}$. We say η' is an element of the subtree rooted at η if η' in \mathcal{T} can be obtained by extending η with a sequence from Σ . For a node η that is not a leaf, the *children* of η are elements η_i such that $\eta_i = \eta \cdot \langle a \rangle$ for $a \in \Sigma$. The height of a tree is defined as the maximum length sequence in it. A leaf has height 1, and every node has a height one more than any of its children. We separately define levels of nodes in a tree for Even and Odd inductively as follows. An *Even level* of a node is 2 if it is a leaf, and it is two more than its children's Even level if it is not a leaf. The Odd level is defined similarly, starting at level 1 for leaves. Note that the height of a tree is at most half of either the Even or Odd level.

We only consider trees such that the children of each node all have the same level. For any node, we usually use the same variable with subscripts to list their children in order i.e., for η , we use η_1, \dots, η_k to denote its first k children

A complete n -ary tree of height $2h$, is the tree where each node has n children and the height of the root is $2h$. For this an n -ary tree, we use $\eta \cdot \langle i \rangle$ to denote the i^{th} child of η . A tree is (n, h) -universal if there is an order preserving injective homomorphism from any tree of height h and with n leaves into it.

Attractor decomposition. A structurally simplest quasi-dominion is an *atomic quasi-dominion*: it is a set $T \cup H$ such that T is an Even attractor to H in $T \cup H$, and all vertex priorities in H are even and larger than all vertex priorities in T . It is a quasi-dominion because if Even uses an attractor strategy in T and an arbitrary strategy in H , then a play that stays in $T \cup H$ forever visits set H infinitely many times and hence it is winning for Even. Consider the following three ways of composing quasi-dominions into structurally more complex ones.

- If Q_1 and Q_2 are quasi-dominions and Q_1 is a trap for Odd in $Q_1 \cup Q_2$, then $Q_1 \cup Q_2$ is a quasi-dominion. We say that $Q_1 \cup Q_2$ is a *sequential composition* of Q_1 and Q_2 .
- If Q is a quasi-dominion, S is an attractor to Q in $Q \cup S$, and Q is a trap for Odd in $Q \cup S$, then $Q \cup S$ is a quasi-dominion. We say that $Q \cup S$ is a *side-attractor composition* of Q and S .
- If Q is a quasi-dominion, T is an attractor to H in $Q \cup (T \cup H)$, and if all vertex priorities in H are even and larger than all vertex priorities in $Q \cup T$, then $Q \cup (T \cup H)$ is a quasi-dominion. We say that $Q \cup (T \cup H)$ is a *top-attractor composition* of Q and $T \cup H$, and that T is a *top attractor* to H .

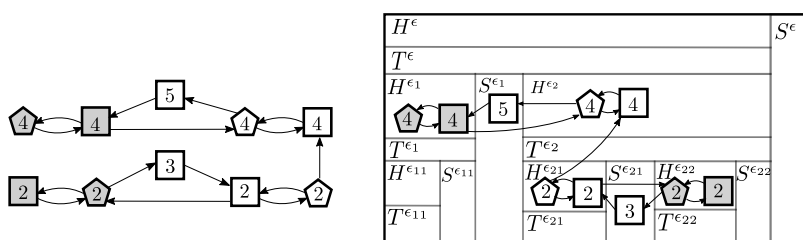
Henceforth, when we say that a quasi-dominion can be obtained by sequential, side-attractor, or top-attractor composition, we implicitly assume that the suitable applicability conditions listed above hold.

Note that an atomic Even quasi-dominion is a special case of the top-attractor composition, in which $Q = \emptyset$. A sequential composition is a quasi-dominion because if Even uses quasi-dominion strategies in Q_1 and Q_2 , then every play that stays in $Q_1 \cup Q_2$ forever, either stays in Q_2 forever, or it eventually stays in Q_1 forever, and hence it is winning for Even. A side-attractor composition is a quasi-dominion because if Even uses quasi-dominion strategy in Q and an attractor strategy in S , then every play that stays in $Q \cup S$ forever, eventually stays in Q forever, and hence it is winning for Even. A top-attractor composition is a quasi-dominion because if Even uses a quasi-dominion strategy in Q , an attractor strategy in T , and an arbitrary strategy in H , then every play that stays in $Q \cup (T \cup H)$ forever, either eventually stays in Q forever, or it visits set H infinitely many times, and hence it is winning for Even.

It is folklore that the three composition operations are complete in the sense that every quasi-dominion can be obtained by a sequence of such composition operations [26, 33]. More specifically, the following concept of an attractor decomposition [7, 20] is a technically convenient normal form, in which the hierarchical structure of quasi-dominions resulting from the composition operations is captured by an ordered tree.

► **Definition 1 (Attractor decomposition).** *An attractor decomposition of a game \mathcal{G} consists of an ordered tree, and for each node ϵ of the tree, three mutually disjoint sets of vertices H^ϵ (high even priority set), T^ϵ (top attractor), and S^ϵ (side attractor) that satisfy the following conditions. Note that these vertices can be potentially empty for some nodes ϵ .*

1. *The root node has some even level, and the children of every node at level k have level $k-2$. For every node ϵ , if its level is k , then all vertex priorities in H^ϵ are k , all vertex priorities in T^ϵ are at most $k-1$, and all vertex priorities in S^ϵ are at most $k+1$.*



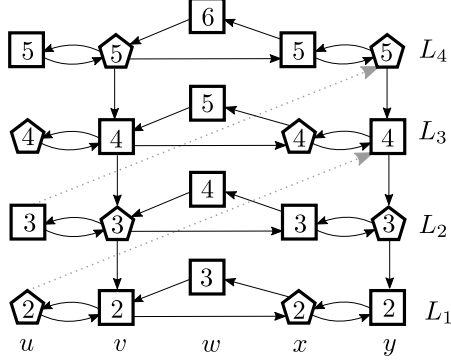
■ **Figure 1** On the left, a game where Even wins from every vertex and on the right, its attractor decomposition.

2. If node ϵ is a leaf, then $T^\epsilon \cup H^\epsilon$ is an atomic quasi-dominion, and quasi-dominion Q^ϵ can be obtained by side-attractor composition of $T^\epsilon \cup H^\epsilon$ and S^ϵ .
3. If ϵ is node that is not a leaf, then quasi-dominion Q^ϵ can be obtained in the following way. Firstly, quasi-dominion P can be obtained by the sequential composition of quasi-dominions Q^η , where η ranges over the children of ϵ in the tree order. Then, quasi-dominion R can be obtained by the top-attractor composition of P and $T^\epsilon \cup H^\epsilon$. Finally, quasi-dominion Q^ϵ can be obtained by the side-attractor composition of R and S^ϵ .

Consider an example of a game and an attractor decomposition described in Figure 1. The priorities are written in the vertices. Player Even owns all square vertices and Odd owns all the pentagons. Some vertices are shaded to be able to identify each vertex in the game uniquely. In this game, Even has a strategy to win from everywhere. An attractor decomposition of this game is depicted to its right. The nodes of the tree are depicted by $\epsilon, \epsilon_1, \epsilon_2, \epsilon_{11}, \epsilon_{21}$ and ϵ_{22} . The corresponding sets are depicted in the figure with some of them being empty. This attractor decomposition satisfies properties 1, 2 and 3 in Definition 1.

McNaughton-Zielonka algorithm. We recall the classical recursive algorithm of McNaughton and Zielonka [26, 33]. However, this is not similar to a description of McNaughton and Zielonka one would find in the wild. It is enhanced to produce attractor decompositions for both the players and also return the winning sets for players Even and Odd. This is accomplished in Algorithm 1 where the attractor decomposition produced is similar to the one defined before. Note that the work of Jurdziński, Morvan and Thejaswini [20] also produce attractor decompositions explicitly for McNaughton and Zielonka.

The algorithm uses two mutually recursive calls, $\text{McNZ}_{\text{Even}}$ and McNZ_{Odd} which takes as input a game \mathcal{G} , the highest priority h . We also further include as input to these calls, two nodes ϵ and ω where ϵ belongs to an Even n -ary tree and ω to an Odd n -ary tree. The respective levels of these nodes in the tree are h and $h + 1$. These trees are used to store the structure of the quasi-dominions obtained in the algorithm thus far. Moreover, we assume that the algorithm has access to the recursive structure of the quasi-dominions computed by it in previous recursive subcalls and can access the sets corresponding to this quasi-dominion with the help of the nodes of the tree. The quasi-dominions computed up until a point in the algorithm are stored as a disjoint partition of sets, based on the underlying complete tree. These disjoint sets are represent with $H_{\text{Even}}^\epsilon, T_{\text{Even}}^\epsilon$ and S_{Even}^ϵ for each node ϵ belonging to the n -ary tree used for Even tree and $H_{\text{Odd}}^\omega, T_{\text{Odd}}^\omega$ and S_{Odd}^ω for ω belonging to the Odd tree. We exclusively use ϵ and its variants for Even trees and ω and its variants for Odd trees to avoid confusion. The statement $S_{\text{Odd}}^{\omega \cdot (i)} \leftarrow A_i' \setminus U_i'$ performs the side attractor composition for Odd and the statements $H_{\text{Even}}^\epsilon \leftarrow H_i$, and $T_{\text{Even}}^\epsilon \leftarrow A_i \setminus H_i$ together perform a top attractor composition for the Even attractor decomposition.



■ **Figure 2** The game \mathcal{H}_4 .

■ **Algorithm 1** The McNaughton-Zielonka Algorithm.

```

procedure McNZEven( $\mathcal{G}, h, \epsilon, \omega$ ):
    if  $h = 0$  then
        | return  $\emptyset$ 
     $\mathcal{G}_1 \leftarrow \mathcal{G}, i = 0$ 
    repeat
         $i \leftarrow i + 1$ 
         $H_i \leftarrow \pi^{-1}(h) \cap \mathcal{G}_i$ 
         $A_i \leftarrow \text{Attr}_{\text{Even}}^{\mathcal{G}_i}(H_i)$ 
         $\mathcal{G}'_i \leftarrow \mathcal{G}_i \setminus A_i$ 
         $U'_i \leftarrow$ 
            McNZOdd( $\mathcal{G}'_i, h - 1, \omega \cdot \langle i \rangle, \epsilon$ )
         $A'_i \leftarrow \text{Attr}_{\text{Odd}}^{\mathcal{G}'_i}(U'_i)$ 
         $S_{\text{Odd}}^{\omega \cdot \langle i \rangle} \leftarrow A'_i \setminus U'_i$ 
         $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus A'_i$ 
    until  $\mathcal{G}_i = \mathcal{G}_{i+1}$ 
     $H_{\text{Even}}^\epsilon \leftarrow H_i$ 
     $T_{\text{Even}}^\epsilon \leftarrow A_i \setminus H_i$ 
    return  $V(\mathcal{G}_i)$ 
    
```

We only describe the Even recursive call since the Odd recursive subcall can be described similarly. The correctness of the above algorithm are well known and we refer an interested reader to several pre-existing works [26, 33, 19, 20] proving the correctness of the algorithm.

Hard examples for McNaughton and Zielonka. The McNaughton-Zielonka algorithm outperforms several other algorithms in practice, but there are several families of games on which it takes exponential time. Some examples are those found in the paper of Friedmann [13], Gzada and Willemse [14] and van Dijk [31], Benerecetti et al [2]. We add to this list, a family of games on which McNaughton-Zielonka makes exponentially many recursive subcalls. We focus our attention in this section to this family and use it as a running example to highlights the exponential complexity of McNaughton-Zielonka and to motivate the idea behind our technique in the following sections.

Consider the following family of games \mathcal{H}_k for each k which we define below. The game \mathcal{H}_k consists of $5k$ vertices, with the highest priority being $k + 2$. For each i at most k , the vertex set consists of 5 vertices, and they are $\{u_i, v_i, w_i, x_i, y_i\}$. We call this set L_i and refer to it as the i^{th} layer of the game. The priority of w_i is $i + 2$ and all the other nodes in L_i have priority $i + 1$. For even values of i , Odd owns v_i, y_i and Even owns u_i, x_i . For odd values of i , this is swapped. The ownership of w_i is irrelevant. The edges within a layers L_i are: $\{(u_i, v_i), (v_i, u_i), (v_i, x_i), (x_i, w_i), (w_i, v_i), (x_i, y_i), (y_i, x_i)\}$. Between layers,

- for each $i \leq k - 2$, there is an edge (u_i, y_{i+2}) ;
- for each $1 < i \leq k$, there are edges (v_i, v_{i-1}) and (y_i, y_{i-1}) .

An example of the game \mathcal{H}_4 is shown in Figure 2, where square vertices are owned by Even and pentagon by Odd. The odd layers in the game are winning for Even, whereas the Even layers are winning for Odd. A strategy witnessing the above of each player is for a player to move to the vertex to their left. More formally, for each player and an appropriate j , the edges (v_j, u_j) and (y_j, x_j) turns out to be a winning strategy in their respective dominions.

► **Lemma 2.** *For the family of games \mathcal{H}_n for $n \geq 1$ Algorithm 1 makes $O(2^n)$ recursive subcalls to $\text{McNZ}_{\text{Even}}$ and McNZ_{Odd}*

We provide the proof for the above in the full version of the paper but restrict ourselves to a discussion to highlight the idea behind the exponential complexity of McNaughton-Zielonka. For an odd value k , the procedure $\text{McNZ}_{\text{Even}}$ on \mathcal{H}_k makes two McNZ_{Odd} subcalls and these are on the subgames $\mathcal{H}_{k-1} \cup \{u_k, v_k, y_k\}$ and $\mathcal{H}_{k-1} \cup \{x_k, y_k\}$ in succession. Notice that these games have a large intersection, which includes \mathcal{H}_{k-1} and the vertex y_k , leading to an exponential complexity of this easy-to-describe algorithm. The first recursive call $\mathcal{H}_{k-1} \cup \{u_k, v_k, y_k\}$ indeed identifies winning sets for this subgame for both players. Moreover, for the Even player, the strategies that are winning in this subgame is in fact winning in \mathcal{H}_k too. Unfortunately, the next recursive subcall promptly discards this information.

It is natural to ask if there is some way we can utilise the progress we make in the first recursive subcall to provide a head start for the following recursive subcalls. Several enhancements of McNaughton-Zielonka exist, all of which attempt to utilise some information from recursive subcalls. A notable few include the Priority promotion algorithms and its variants [1, 3] along with the Tangle learning algorithms by van Dijk [31], which all perform well in practice. The key idea behind these algorithms is to identify quasi-dominions in their recursive subcalls and increase the quasi-dominions obtained so far carefully. Another idea was to remember the strategy obtained in the recursive subcall. In a recent work by Lapauw, Bruynooghe, and Denecker [22], they show that by remembering and modifying strategies obtained recursively in a calculated manner, they too obtained faster practical performances. But all of these have worst case exponential running time comparable to McNaughton-Zielonka, as these heuristics do not lead to a provable increase in the running time of either of these algorithms.

We propose a technique in the next section, which remembers some information obtained from the structure of the attractor decompositions computed from previous recursive subcalls. This turns out to provide a quadratic gain in the worst case runtime complexity.

3 Making McNaughton-Zielonka faster with some memory

The crucial object that we will be dealing with for the rest of this paper are *decompositions*, which forms the central theme of this section. We define this object as a generalisation of an attractor decomposition. With the help of our definition, we carefully modify the classical algorithm to make at most $O\left(\binom{n}{h}^{h/2}\right)$ many recursive calls. This is comparable to the runtime of the first progress measure algorithm by Jurdziński [17]. For the proof of correctness and runtime of the algorithm, we need the machinery that we develop in Section 4. This mathematical tool-kit helps accurately pin point the guarantees of the algorithm and also analyse the running time. We however state these guarantees in this Section and with these guarantees use them to demonstrate how our algorithm works faster on families of games which are hard for several attractor based algorithms. We show that this modification to McNaughton-Zielonka makes it run in polynomial time for two specific families.

A relaxation of attractor decomposition

Recall the procedure $\text{McNZ}_{\text{Even}}$ on example \mathcal{H}_k from Section 2. This procedure makes two recursive subcalls of McNZ_{Odd} to subgames each containing \mathcal{H}_{k-1} . Notice that although McNZ_{Odd} returned an attractor decomposition on the first iteration, we discarded the Even attractor decomposition obtained from this recursive subcall until an empty Odd dominion was returned by the recursive subcall. In fact, this algorithm on any game repeatedly discards this information until U'_i returned is empty.

Indeed, this wasteful discarding of attractor decompositions seems necessary to prove correctness at first sight. On removing vertices from an Even attractor decomposition it might no longer satisfy all the properties of an attractor decomposition. Since after each recursive subcall, we remove a non-empty Odd attractor A'_i from it, it seems natural that this information must now be discarded. To circumvent this wastage, we propose to store an object that loosely resembles an attractor decomposition but with the requirements about quasi-dominions and attractors relaxed. Below, we define an Even decomposition to closely resemble our definition of attractor decomposition from the previous section.

► **Definition 3 (Decomposition).** *A decomposition consists of an ordered tree, and for each node ϵ of the tree, three mutually disjoint sets of vertices H^ϵ (high even priority set), T^ϵ (top set), and S^ϵ (side set) that satisfy only the following condition.*

- *The root node has some even level, and the children of every node at level k have level $k-2$. For every node t , if its level is k , then all vertex priorities in H^ϵ are k , all vertex priorities in T^ϵ are at most $k-1$, and all vertex priorities in S^ϵ are at most $k+1$.*

Notice that this definition is more general than an attractor decomposition. A decomposition which satisfies the attractor composition properties (items 2 and 3) in Definition 1 is an attractor decomposition. More importantly, since a decomposition no longer needs to satisfy properties about traps and attractors, on restricting it to a subset of vertices, it still remains a decomposition. This helps us maintain this structure even after the algorithm removes some vertices from it.

A faster algorithm

The modification we propose for McNaughton-Zielonka is described in Algorithm 2. This algorithm, at a high level resembles Algorithm 1. The significant difference is that it starts with a decomposition and modifies it by calling recursive subcalls until this decomposition is an attractor decomposition.

For a parity game \mathcal{G} , the algorithm maintains two decompositions with n -ary trees for both the players. This is done globally and refers to the sets associated to these decompositions using the nodes of the tree. We refer to decompositions $\mathcal{D}_{\text{Even}}^\epsilon$ or $\mathcal{D}_{\text{Odd}}^\omega$ for ϵ and ω which are nodes in the Even and Odd tree respectively. Note that although we use $\mathcal{D}_{\text{Even}}^\epsilon$ or $\mathcal{D}_{\text{Odd}}^\omega$, these are two distinct decompositions. Moreover, we use ϵ along with Even in the subscript, along with its variations like ϵ' , ϵ_i , \dots for nodes in the Even tree and similarly for Odd with ω , this enables us to refer to both the Even and Odd decompositions without confusion.

We call the partitions in these decomposition H_{Even}^ϵ , T_{Even}^ϵ and S_{Even}^ϵ for ϵ , a node in the Even tree, and H_{Odd}^ω , T_{Odd}^ω and S_{Odd}^ω for ω in the Odd tree. We use $[\mathcal{D}_{\text{Even}}^\epsilon]$ to denote the set of vertices associated with some node in the subtree of the tree rooted at ϵ but without the vertices in S_{Even}^ϵ . More formally, for a decomposition $\mathcal{D}_{\text{Even}}^\epsilon$, we define $[\mathcal{D}_{\text{Even}}^\epsilon]$ inductively, where

- $[\mathcal{D}_{\text{Even}}^\epsilon]$ consists of $H^\epsilon \cup T^\epsilon$ if ϵ is a leaf;
- $[\mathcal{D}_{\text{Even}}^\epsilon]$ consists of $H_{\text{Even}}^\epsilon \cup T_{\text{Even}}^\epsilon$ along with all vertices in $[\mathcal{D}_{\text{Even}}^{\epsilon_i}] \cup S_{\text{Even}}^{\epsilon_i}$ for ϵ_i ranging over the children of ϵ .

The procedure `McNZFastEven` in Algorithm 2 works using decompositions $\mathcal{D}_{\text{Even}}^\epsilon$ and $\mathcal{D}_{\text{Odd}}^\omega$ on a subgame \mathcal{G} . We modify these decompositions with operations like “Set” and “Move”. We only give an intuition of the operations here, but we make these precise in the appendix of the paper. The highest priority in the game \mathcal{G} is at most the level of Even tree’s root: ϵ . Moreover, the level of Odd tree’s root ω is one more than the level of ϵ .

■ **Algorithm 2** McNaughton and Zielonka Algorithm with memory.

```

procedure McNZFastEven( $\mathcal{G}, h, \epsilon, \omega$ ):
  if  $\mathcal{D}_{\text{Even}}^\epsilon$  restricted to  $\mathcal{G}$  is an
    attractor decomposition then
    | Set  $S_{\text{Odd}}^\omega$  to  $V(\mathcal{G})$ 
    | return  $V(\mathcal{G})$ 
  else if  $\mathcal{D}_{\text{Odd}}^\omega$  restricted to  $\mathcal{G}$  is an
    attractor decomposition then
    | Set  $S_{\text{Even}}^\epsilon$  to  $V(\mathcal{G})$ 
    | return  $\emptyset$ 
  else
    |  $\mathcal{G}_1 \leftarrow \mathcal{G}; i = 0$ 
    | repeat
    |   |  $i \leftarrow i + 1$ 
    |   |  $H_i \leftarrow \pi^{-1}(h) \cap \mathcal{G}_i$ 
    |   |  $T_i \leftarrow \text{Attr}_{\text{Even}}^{\mathcal{G}_i}(H_i)$ 
    |   | Set  $T_{\text{Even}}^\epsilon$  to  $T_i \setminus H_i$ 
    |   |  $S_i \leftarrow$ 
    |   |    $\text{Attr}_{\text{Even}}^{\mathcal{G}_i}(\mathcal{G}_i \setminus [\mathcal{D}_{\text{Odd}}^{\omega \cdot \langle i \rangle}])$ 
    |   | Move  $S_i$  to at least  $S_{\text{Odd}}^{\omega \cdot \langle i \rangle}$ 
    |   |  $\mathcal{G}'_i \leftarrow (\mathcal{G}_i \setminus S_i)$ 
    |   |  $U'_i \leftarrow$ 
    |   |    $\text{McNZFast}_{\text{Odd}}(\mathcal{G}'_i, h - 1, \epsilon, \omega \cdot \langle i \rangle)$ 
    |   |
    |   |  $S'_i \leftarrow \text{Attr}_{\text{Odd}}^{\mathcal{G}'_i}(U'_i)$ 
    |   | Set  $S_{\text{Odd}}^{\omega \cdot \langle i \rangle}$  to  $S'_i \setminus U'_i$ 
    |   | Move  $S'_i$  to at least  $S_{\text{Even}}^\epsilon$ 
    |   |  $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus S'_i$ 
    | until  $\mathcal{D}_{\text{Even}}^\epsilon$  restricted to  $\mathcal{G}_{i+1}$  is
    an attractor decomposition
    | return  $V(\mathcal{G}_i)$ 

```

■ **Algorithm 3** Universal Attractor Decomposition algorithm with memory.

```

procedure UnivFastEven( $\mathcal{G}, h, \epsilon, \omega$ ):
  if  $\mathcal{D}_{\text{Even}}^\epsilon$  is an attractor
    decomposition for  $\mathcal{G}$  then
    | Set  $S_{\text{Odd}}^\omega$  to  $V(\mathcal{G})$ 
    | return  $V(\mathcal{G})$ 
  else if  $\mathcal{D}_{\text{Odd}}^\omega$  is an attractor
    decomposition for  $\mathcal{G}$  then
    | Set  $S_{\text{Even}}^\epsilon$  to  $V(\mathcal{G})$ 
    | return  $\emptyset$ 
  else
    |  $\mathcal{G}_1 \leftarrow \mathcal{G}$ 
    | Let  $\omega_1, \dots, \omega_k$  be children of  $\omega$  in
    the Odd tree
    | for  $i \leftarrow 1$  to  $k$  do
    |   |  $H_i \leftarrow \pi^{-1}(h) \cap \mathcal{G}_i$ 
    |   |  $T_i \leftarrow \text{Attr}_{\text{Even}}^{\mathcal{G}_i}(H_i)$ 
    |   | Set  $T_{\text{Even}}^\epsilon$  to  $T_i \setminus H_i$ 
    |   |  $S_i \leftarrow \text{Attr}_{\text{Even}}^{\mathcal{G}_i}(\mathcal{G}_i \setminus [\mathcal{D}_{\text{Odd}}^{\omega_i}])$ 
    |   | Move  $S_i$  to at least  $S_{\text{Odd}}^{\omega_i}$ 
    |   |  $\mathcal{G}'_i \leftarrow (\mathcal{G}_i \setminus S_i)$ 
    |   |  $U'_i \leftarrow$ 
    |   |    $\text{UnivFast}_{\text{Odd}}(\mathcal{G}'_i, h - 1, \epsilon, \omega_i)$ 
    |   |
    |   |  $S'_i \leftarrow \text{Attr}_{\text{Odd}}^{\mathcal{G}'_i}(U'_i)$ 
    |   | Set  $S_{\text{Odd}}^{\omega_i}$  to  $S'_i \setminus U'_i$ 
    |   | Move  $S'_i$  to at least  $S_{\text{Even}}^\epsilon$ 
    |   |  $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus S'_i$ 
    | return  $V(\mathcal{G}_{i+1})$ 

```

If the decompositions computed so far already forms an attractor decomposition for either player, the algorithm stops and returns the corresponding set, since having an attractor decomposition implies that we have a witness of winning for either player in the current subgame. We claim that one can check if a decomposition is an attractor decomposition efficiently, in polynomial time in the number of vertices of the game rather than the size of the tree, but postpone the details on how until the next section.

On the other hand, if both players only have decompositions that are not attractor decompositions on the current subset of vertices, we first compute the Even attractor to the top priority in the current subgame, closely mirroring Algorithm 1. However, we deviate from McNaughton-Zielonka, by updating this information by modifying the partition T_{Even}^ϵ of the Even decomposition.

The line Set T_{Even}^ϵ to $T_i \setminus H_i$ results in the current decomposition of even being modified so that the “top set” T_{Even}^ϵ now contains exactly the vertices $T_i \setminus H_i$, which could be attracted to the set of vertices of top priority H_i in the current subgame. Doing this updates requires modification of the decomposition to relocate the other vertices that were previously at T_{Even}^ϵ to sets associated to the children of ϵ .

In Algorithm 1, the next recursive subcall would work on the complement of the attractor set T_i . However, for this algorithm, the next recursive subcall at one level lower, we consider a subset of the set $\left[\mathcal{D}_{\text{Odd}}^{\omega \cdot \langle i \rangle}\right]$. Since this need not be a subgame, the procedure computes the Even attractor to all vertices not in this set. The complement of this Even attractor results in S_i , a trap for Even. Intuitively, the next line: Move S_i to at least $S_{\text{Odd}}^{\omega \cdot \langle i \rangle}$ modifies the decomposition such that in the new decomposition obtained, the vertices in S_i are no longer in the sets at $\left[\mathcal{D}_{\text{Odd}}^{\omega \cdot \langle i \rangle}\right]$. Instead, these vertices are such that assigned the vertices in S_i to the sets corresponding to either move these vertices to $S_{\text{Odd}}^{\omega \cdot \langle i \rangle}$ in the Odd decomposition. The other partitions in the decomposition are left unchanged.

We then perform an Odd recursive subcall, by calling the procedure $\text{McNZFast}_{\text{Odd}}$ on \mathcal{G}'_i , the trap for Even obtained above. After the Odd recursive subcall, which returns the Odd winning set U'_i in the subgame \mathcal{G}'_i , we compute the Odd attractor to this set which is S'_i . Since S'_i is the set of vertices from which Odd has a strategy to reach U'_i , we adjust the Odd decomposition such that the side set $S_{\text{Odd}}^{\omega \cdot \langle i \rangle}$ is exactly $S'_i \setminus U'_i$ and all vertices in S'_i are then relocated to S_{Even}^ϵ for the Even decomposition. This is captured by the line Move S'_i to at least S_{Even}^ϵ . This is done similarly to the above move operator, where all vertices in S'_i in the newly obtained decomposition

Now, all we need to explain is how we initialise these decompositions for the algorithm. The following definition seem technical, although intuitively, we start with an “optimistic” decomposition, which starts with the simplest structure of a decomposition for both Odd and Even. We call the *initial Even decomposition* for the game \mathcal{G} with highest priority h , where the Even node at level h is ϵ , and the Odd node at level $h + 1$ is ω , as the following:

- all vertices of priority h are at H_{Even}^ϵ for Even.
- all vertices with priority strictly smaller than $h - 1$ are at T_{Even}^ϵ for Even.

Similarly, the *initial Odd decomposition* is defined as

- all vertices of priority h are at S_{Odd}^ω for Odd.
- all vertices of priority exactly $h - 1$ are at $H_{\text{Odd}}^{\omega_1}$ and all vertices of priority strictly smaller than $h - 1$ are at $T_{\text{Odd}}^{\omega_1}$ for Odd

With this, we can state the guarantees our modification provides.

► **Theorem 4.** *Let \mathcal{G} be a parity game with priorities no larger than an even number h , for two n -ary trees of height $h/2$ and $h/2 + 1$ with roots ϵ and ω respectively. Procedure $\text{McNZFast}_{\text{Even}}(\mathcal{G}, h, \epsilon, \omega)$, with the underlying decomposition being the initial Even and Odd decomposition of \mathcal{G} into these trees output the winning set of Even, and the decomposition computed is the same as the one computed by procedure $\text{McNZ}_{\text{Even}}$ in Algorithm 1.*

The following statement proves runtime of this algorithm, and shows the quadratic improvement we gain compared to McNaughton-Zielonka, and comparable to the the small progress measure algorithm by Jurdziński [17].

► **Lemma 5.** *On a game \mathcal{G} with n vertices with priority at most h , the number of recursive subcalls by either $\text{McNZFast}_{\text{Even}}$ or $\text{McNZFast}_{\text{Odd}}$ is at most the product of a polynomial in n and $O\left(\frac{n}{h}\right)^{\lceil h/2 \rceil}$.*

Examples of faster termination

We will demonstrate the algorithm on two examples in this subsection to understand this. These two examples are going to be the family of games \mathcal{H}_k introduced in Section 2 and the family of games, we will call \mathcal{F}_k , which was introduced by Friedmann [13].

We recall that in his work, Friedmann had provided a family of games on which McNaughton-Zielonka takes exponential time [13]. We will call this family of examples \mathcal{F}_k for $k \in \mathbb{N}$ but will not describe this family in detail, and instead refer the reader to the work Friedman [13]. Friedmann, showed in Theorem 4.3, of [13] that on the game \mathcal{F}_k , the algorithm makes F_k recursive subcalls where F_k denotes the n^{th} Fibonacci number, bounded by approximately $(1.618)^k$. This is smaller than the 2^n time for the previous example we have shown, however, one can remark that because of the structure of the game, it makes it difficult for several algorithms to solve this game. Indeed, the priority promotion algorithms without any enhancements such as memoisation or delayed promotion also takes exponential time on these games. This exponential behaviour is due to their “rests” performed. We show that our algorithm solves this family of games provided by Friedmann in polynomial time as well as our running example of \mathcal{H}_k in the next two lemmas.

► **Lemma 6.** *Algorithm 2 when initialised with the trivial decomposition for both players solves the family of games \mathcal{H}_n in time that is polynomial in n .*

► **Lemma 7.** *Algorithm 2 when initialised with the trivial decomposition for both players solves the family of games \mathcal{F}_n , introduced by Friedmann [13] in time that is polynomial in n .*

The key idea behind both these proofs is that we do enough work by maintaining a decomposition in an initial recursive subcall. In the future recursive subcalls, when this decomposition is our starting point, the algorithm needs to do at most polynomial amount of processing in the above family of games by showing that they fit one of the following criterion:

- already have a witness for winning in the form of an attractor decomposition from a previous recursive subcall (To prove polynomial termination of \mathcal{H}_k in Lemma 6);
- although the current decomposition of an Even dominion, is the initial decomposition, the decomposition maintained for Odd is robust enough to conclude quickly that it is losing for Odd, and therefore winning for Even; (used in the proof of Lemma 7).
- each of the next several recursive subcalls made are on a significantly smaller subgames due to the structure of the available decompositions. Some of these subcalls might even turn out to be empty (Lemma 6 used to prove fast termination of \mathcal{F}_k). This happens dually with the above mentioned phenomenon;
- starting from a specific decomposition for the game, we only require polynomial time using our algorithm to arrive at an attractor decomposition (used in the proof of Lemma 6).

The exact details of these proof requires us to identify specific subgames in the recursive calls and are available in the full version of the paper. Although the proof of both Lemmas rely on induction which in turn depends on the regularity of the subgames in recursive calls, this is not essential for our algorithm to terminate faster and is just an easier proof mechanism. This distinguishes our algorithm from targeted tricks aimed at solving specific families of games faster.

4 Using smaller trees

Attractor decompositions function as a proof of winning for parity games, and can be defined on the recursive structure of trees. Universal trees have been key to all known quasi-polynomial algorithms [6]. The use of small universal trees in the attractor based quasi-polynomial algorithms of Lehtinen, Parys, Schewe and Wojtczak [25] was highlighted in works of Jurdziński, Morvan and Thejaswini [20]. Their work also captured the recursive structure of these witnesses with the help of trees. Moreover, they generalise their algorithm to work on arbitrary trees, and show stronger guarantees. As a corollary, they obtain that

using two universal trees in their algorithm gives a correct algorithm for solving parity games. However, the theoretical complexity of these algorithms do not match the run-time of the state of the art. For two trees $\mathcal{T}^{\text{Even}}$ and \mathcal{T}^{Odd} , these algorithms take time proportional to the interleaving of the two trees. This object has size equal to the product of the trees, $|\mathcal{T}^{\text{Even}}| \cdot |\mathcal{T}^{\text{Odd}}|$, as opposed to the algorithms with state of the art [18] worst case complexity of time which is linear in the size of each tree. In fact, in the journal version of the paper of Lehtinen, Parys, Schewe and Wojtczak [25], they emphasise in their introduction that their algorithm has a gap when compared to other quasi-polynomial algorithms. We describe how our technique can be applied to the algorithms of Lehtinen, Parys, Schewe and Wojtczak [25] as well as Jurdziński, Morvan and Thejaswini [20] in this section which would closes this gap. We also achieve this by extending our results to arbitrary trees, of which their algorithms form specific instances. For a detailed report on which universal trees correspond to which algorithm, we refer the reader to the work of Jurdziński, Morvan and Thejaswini [20].

Algorithm using arbitrary trees

To apply the technique of remembering decompositions for quasi-polynomial versions of attractor based algorithm, we need to restrict the exponential branching in the previous algorithm. In our algorithm, we bound the branching in the algorithm to inter-leavings of any two arbitrary trees with minimal modification from our previous algorithms. Instantiating these trees to the universal trees underlying the algorithm of Parys [27], or Lehtinen, Schewe, and Wojtczak [24] results in a speed up compared to these respective algorithms.

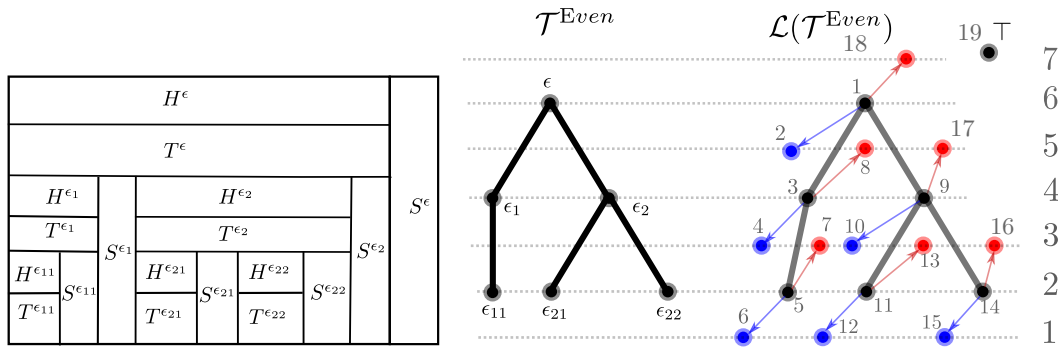
The version of our algorithm which uses any two arbitrary trees contains two mutually recursive procedures $\text{UnivFast}_{\text{Even}}$ and $\text{UnivFast}_{\text{Odd}}$ which take as input a game \mathcal{G} , the highest priority h in the game, and two nodes ϵ and ω from \mathcal{T}^{Odd} and $\mathcal{T}^{\text{Even}}$. The nodes ϵ and ω have level h and $h + 1$ respectively in their trees for the Even procedure and vice versa for the Odd procedure. The underlying trees are not passed in a recursive subcall as we assume they are a part of the algorithm and can be accessed globally. Other than the arbitrary trees, this algorithm deviates slightly from our previous algorithms in one way. The main loop in this algorithm, whose branching was earlier (virtually) unbounded, is now determined by the trees \mathcal{T}^{Odd} and $\mathcal{T}^{\text{Even}}$. The pseudo-code for the Even recursive subcall is specified in Algorithm 3 and is given next to the pseudo-code of Algorithm 2 for ease of comparison.

Decompositions and labels

To prove correctness of the algorithms as well as analyse their runtime more carefully, we offer an alternate way of viewing decompositions. Instead of a partition in the shape of a tree, we can equivalently view it as a map to a specific tree. We define such trees, which we call *leafy trees*. We then show that decompositions are nothing but maps into such specific ordered trees. This ordering on the tree induces a natural ordering on the set of all decompositions.

This alternate view also helps us argue correctness and termination using monovariance of the decomposition maintained throughout the algorithm.

Leafy tree. Let \mathcal{T} be an ordered tree with a root at level h , an even value. We call the leafy tree of a tree \mathcal{T} , denoted by $\mathcal{L}(\mathcal{T})$ to be an ordered set which contains three copies of each element in the tree \mathcal{T} and another element \top . We define this more formally below.



■ **Figure 3** From left to right: A decomposition set into the tree $\mathcal{T}^{\text{Even}}$, tree $\mathcal{T}^{\text{Even}}$ and its corresponding leafy tree with its order.

► **Definition 8.** Given a tree \mathcal{T} , we introduce two additional nodes for each node $\eta \in \mathcal{T}$, which we will call η^T and η^S to denote the nodes corresponding to top and side nodes respectively. Other than this, we add an element \top . We say, that

$$\mathcal{L}(\mathcal{T}) = \mathcal{T} \cup \{\eta^T \mid \eta \in \mathcal{T}\} \cup \{\eta^S \mid \eta \in \mathcal{T}\} \cup \{\top\}.$$

We sometimes refer to the nodes from \mathcal{T} as a node in the skeleton of the leafy tree. The order of elements in $\mathcal{L}(\mathcal{T})$ is as follows: $\eta \in \mathcal{T}$ and for $\eta' \in \mathcal{L}(\mathcal{T}')$, where \mathcal{T}' is a strict subtree of the tree rooted at η , we have $\eta < \eta^T < \eta' < \eta^S$. Moreover, the order on \mathcal{T} is inherited by $\mathcal{L}(\mathcal{T})$. We also have $\eta < \top$, for any $\eta \in \mathcal{L}(\mathcal{T})$, which is not \top . Note that the above conditions ensure that $\mathcal{L}(\mathcal{T})$ is a total order and the smallest element strictly larger than η is well defined. We define level of η^S to be one more than the level of η and the level of η^T to be one less.

For a pictorial representation of a leafy tree, look at Figure 3. Here, the Leafy tree of the tree $\mathcal{T}^{\text{Even}}$ with root ϵ , and its children ϵ_1, ϵ_2 and leaves $\epsilon_{11}, \epsilon_{21}$, and ϵ_{22} . In $\mathcal{L}(\mathcal{T}^{\text{Even}})$, we denote elements according to their order, where the elements from $\mathcal{T}^{\text{Even}}$ are the black nodes, η^T with blue and η^S with red. Their level can be inferred by the dotted lines. Also observe a decomposition into the same tree $\mathcal{T}^{\text{Even}}$ placed next to the leafy tree in the picture. We show that a map into a leafy tree can be obtained from the following decomposition.

Labelling. We define a labelling for Even into \mathcal{T} as a map from the vertices of a parity game \mathcal{G} to $\mathcal{L}(\mathcal{T})$, which obeys the following conditions:

- if a vertex is mapped to the skeleton of the leafy tree: $\mathcal{T} \subset \mathcal{L}(\mathcal{T})$, then its priority is the same as the level of the node;
- if a vertex is mapped to “leafy vertex”, i.e. η^S or η^T then its priority is at most the level of the leafy vertex.

There are no restrictions on elements mapped to \top . One could also equivalently define a labelling for Odd by replacing Even with Odd. When we refer to labellings in the rest of the section, we refer to Even labellings.

Given two labellings for Even, λ_1 and λ_2 into tree \mathcal{T} which are maps from V to $\mathcal{L}(\mathcal{T})$, we say that the order on the elements of $\mathcal{L}(\mathcal{T})$ extend to give a partial order on the labelling. More formally, $\lambda_1 \sqsubseteq \lambda_2$ if and only if for all $v \in V$, $\lambda_1(v) \leq \lambda_2(v)$.

Indeed, the above definition corresponds to a decomposition naturally by the following proposition.

- **Proposition 9.** *For a parity game \mathcal{G} , with priorities that do not exceed $h + 1$ and a tree \mathcal{T} of height $h/2$,*
- *for a labelling λ from \mathcal{G} to $\mathcal{L}(\mathcal{T})$, there is a canonical decomposition \mathcal{D}_λ to the tree \mathcal{T} ;*
 - *for a decomposition \mathcal{D} into \mathcal{T} , there is a canonical labelling $\lambda_{\mathcal{D}}$ which maps $V(\mathcal{G})$ to $\mathcal{L}(\mathcal{T})$.*

This allows us to use labellings and decompositions interchangeably, as they have a one to one correspondence with each other. It also induces a partial order \sqsubseteq on decompositions, which is obtained by the partial order on labellings. We now pinpoint when the decomposition corresponding to a labelling forms an attractor decomposition. We say that a vertex u is *valid* in a labelling λ if $\lambda(u) = \top$ or:

- $\lambda(u) \in \mathcal{T}$ and in one step, Even can ensure that she reaches a vertex v such that $\lambda(v) < \lambda(u)^S$.
- $\lambda(u) = t^T$ or $\lambda(u) = t^S$ and there is a reachability strategy from u to vertices in $\{v \mid \lambda(v) < t\}$ without visiting any vertex v in the path where $\lambda(v) > t$.

In the lemma below, we show that validity of all vertices also corresponds to a witness of winning.

- **Lemma 10.** *Given an Even labelling λ , there is a winning strategy for Even from all vertices u such that $\lambda(u) \neq \top$ iff all vertices are valid.*

Not only do such labellings correspond to dominions, the following proposition shows a stronger statement that the decomposition \mathcal{D}_λ corresponding to such a labelling λ is an attractor decomposition exactly if and only if every vertex is valid in λ .

- **Proposition 11.** *In a parity game \mathcal{G} ,*
- *an Even decomposition \mathcal{D} to a tree \mathcal{T} is also an attractor decomposition if the corresponding labelling $\lambda_{\mathcal{D}}$ which maps $V(\mathcal{G})$ to $\mathcal{L}(\mathcal{T})$ is valid for all vertices;*
 - *if a labelling λ is valid at all vertices, then the corresponding \mathcal{D}_λ is an attractor decomposition.*

As defined in these previous works [7, 20, 8], given a dominion, there could be several attractor decompositions for it. Each of these attractor decompositions could correspond to different trees. In our definition, instead of obtaining the trees from a given attractor decomposition, we view it as a map into a fixed tree. The proposition below helps us show that attractor decompositions defined as labellings is robust.

- **Proposition 12.** *Consider a parity game \mathcal{G} with two labellings λ_1 and λ_2 from the vertices to $\mathcal{L}(\mathcal{T})$ for some tree \mathcal{T} . Let $\lambda_1 \sqsubseteq \lambda_2$ and $u \in V(\mathcal{G})$ such that $\lambda_1(u) = \lambda_2(u)$. If u is valid in λ_2 , then it is also valid in λ_1 .*

It can be shown as a corollary of the above proposition, that taking the point-wise minimum of two attractor decompositions is also an attractor decomposition. An interesting consequence of this stated corollary of Proposition 12 is the following lemma, which notes that the set of all attractor decompositions forms a lattice.

- **Lemma 13.** *Given a game \mathcal{G} and a tree \mathcal{T} , the set of all labellings from $V(\mathcal{G})$ to $\mathcal{L}(\mathcal{T})$ which are also attractor decompositions form a lattice, with the \sqsubseteq order. More specifically, it has a unique maximal and minimal element.*

The meet operator is well defined, which makes the set of attractor decompositions a finite semi-lattice. Since there is a unique maximal element, the trivial labelling which maps all elements to \top , this structure forms a lattice. We will focus on this unique minimal attractor decomposition, which plays a key role in our proofs.

Correctness and runtime

With this dual view of an attractor decomposition, we are equipped to prove the correctness and runtime of our algorithms.

Correctness

We prove correctness by proving Theorem 4, which states that on two input trees $\mathcal{T}^{\text{Even}}$ and \mathcal{T}^{Odd} , if there is a dominion of Even that has an attractor decomposition for the tree $\mathcal{T}_{\text{Even}}$, then the decomposition returned contains all vertices in that dominion. Additionally, it contains no vertices from any Odd dominions that has an attractor decomposition for the tree \mathcal{T}_{Odd} . Indeed, if the trees are large enough, and winning set for both Even and Odd have an attractor decomposition into $\mathcal{T}^{\text{Even}}$ and \mathcal{T}^{Odd} respectively, then the sets D_{Odd} and D_{Even} correspond to winning sets exactly and the algorithm exactly returns the winning sets. On smaller trees that cannot include the entire dominion, the set returned in itself is not a winning set, but it partitions the winning dominions captured by these trees. Therefore, this theorem can be seen as a generalisation of the dominion separation theorem (Theorem 17) in [20].

► **Theorem 14.** *Let \mathcal{G} be a parity game with priorities no larger than an even number h , and let $\mathcal{T}^{\text{Even}}$ and \mathcal{T}^{Odd} be two trees of with roots ϵ and ω respectively. Let D_{Even} be the largest Even dominion in \mathcal{G} that has an attractor decomposition into $\mathcal{T}^{\text{Even}}$ and similarly D_{Odd} , the largest Odd dominion that has an attractor decomposition into \mathcal{T}^{Odd} . Procedure $\text{UnivFast}_{\text{Even}}(\mathcal{G}, h, \epsilon, \omega)$, with the decomposition being the initial Even and Odd decomposition of \mathcal{G} outputs a set of vertices W such that $D_{\text{Even}} \subseteq W \subseteq V(\mathcal{G} \setminus D_{\text{Odd}})$.*

We prove the above theorem by showing the following stronger statements. We show that for the smallest attractor decompositions \mathcal{A}^ϵ and \mathcal{A}^ω with respect to the corresponding trees, the algorithm maintains the following invariants for the decompositions:

- 1 $\mathcal{D}_{\text{Even}}^\epsilon \sqsubseteq \mathcal{A}_{\text{Even}}^\epsilon$ and $\mathcal{D}_{\text{Odd}}^\omega \sqsubseteq \mathcal{A}_{\text{Odd}}^\omega$;
- 2 At the end of a recursive subcall of $\text{UnivFast}_{\text{Even}}(\mathcal{G}, h, \epsilon, \omega)$, the decompositions at the end of the recursive call are such that $[\mathcal{D}_{\text{Even}}^\epsilon] \cap [\mathcal{D}_{\text{Odd}}^\omega]$ is empty.

The essence of the proof boils down to showing that each operation performed on the decompositions, increases the decomposition without overtaking the smallest attractor decomposition. We achieve this by using the ordering on labellings developed earlier in this section. Additionally, we prove Theorem 4 with an extra invariant that if the trees $\mathcal{T}^{\text{Even}}$ and \mathcal{T}^{Odd} are complete n -ary trees, then $\mathcal{D}_{\text{Even}}^\epsilon = \mathcal{A}_{\text{Even}}^\epsilon$ and $\mathcal{D}_{\text{Odd}}^\omega = \mathcal{A}_{\text{Odd}}^\omega$. This automatically gives us the proof of correctness for Algorithm 2 and that it provides an attractor decomposition.

A notable observation on the proof of Theorem 4 is that these two invariants mentioned hold true for all three of the Algorithms stated in the paper until now, including McNaughton-Zielonka. However, it is only with complete trees that we can prove that both Algorithm 1 and Algorithm 2 produce the minimal attractor decompositions. This shows that McNaughton and Zielonka and its variants provide a winning strategy on termination, whereas the quasi-polynomial algorithms only produce a partition between the winning and losing vertices, with no strategy for each player. We make precise what guarantees one can achieve using our techniques. We also show a quadratically faster termination for Algorithm 2 and Algorithm 3 than their counterparts which do not maintain a decomposition.

Runtime

We show how our algorithm runs in time that is at most linear in the size of each of the tree, a significant reduction from other attractor based algorithms with a quadratic dependence.

► **Theorem 15.** *Consider a game \mathcal{G} with n vertices and priority at most h . Procedure $\text{UnivFast}_{\text{Even}}$ (resp. $\text{UnivFast}_{\text{Odd}}$) with trees \mathcal{T}_{Odd} and $\mathcal{T}_{\text{Even}}$ makes $O(n^c \cdot \max(|\mathcal{T}_{\text{Odd}}|, |\mathcal{T}_{\text{Even}}|))$ many recursive subcalls to $\text{UnivFast}_{\text{Even}}$ or $\text{UnivFast}_{\text{Odd}}$ for a constant c . The time taken to perform each operations outside of the recursive subcalls is a polynomial in n , thus making the run-time of this algorithm $O(n^d \cdot \max(|\mathcal{T}_{\text{Odd}}|, |\mathcal{T}_{\text{Even}}|))$ for a constant d .*

Our most significant change that contributes to an improvement in the theoretical bound of the running time is that we maintain the decompositions from previous recursive calls. This is crucial in the proof and helps us argue that our modified algorithm does not take too long before there is an increase at least one of our decompositions. We would however like to emphasise that not all implementations of the algorithm would have the claimed run-time, but with carefully designed data structures, the time taken outside a recursive call is at most polynomial. One such implementation would require a data structure which stores each decomposition as a labelling. This labelling is however represented by only maintaining elements in the support of this labelling instead of the whole tree. Since there are at most n vertices, this support is significantly smaller than the size of a potentially quasi-polynomial or exponentially-sized tree. The attractor computations and the respective Set and Move operations performed in the algorithms take time proportional to a polynomial in n , assuming that we have either oracle access or polynomial-time access to queries such as: next sibling of node, parent of a node or child of a node.

5 Outlook

We believe that our technique can be applied to other attractor-based algorithms that were inspired by the McNaughton-Zielonka algorithm [1, 22], but elaborating this in detail is beyond the scope of this paper. The methodology to follow would be analogous to ours: first make explicit how these algorithms are incrementally building attractor decompositions, and then use the decompositions for both players obtained from recursive calls to reduce the sizes of subgames in further recursive calls. Even implementing just the first step of this methodology seems worthwhile: doing so on arbitrary ordered trees, like we did in Section 4, would allow to obtain a generic priority promotion algorithm, which could be made straightforwardly quasi-polynomial by plugging in small universal trees [18, 8], hence generalizing and streamlining the first quasi-polynomial priority promotion algorithm [3].

A weakness of all quasi-polynomial symmetric attractor-based algorithms – including ours – is that they may output correct winning sets, but without constructing winning strategies. This is a major shortcoming in the context of synthesis, where winning strategies correspond to the desired controllers. We argue that our technique, which is based on computing decompositions that are under-approximations of the least attractor decompositions, allows to tackle this weakness with a modest additional computational cost. If the algorithm terminates with a decomposition that is not an attractor decomposition, then the decomposition obtained can serve as a starting point to make further progress. We can run an algorithm for each player that repeatedly increases the decomposition in the underlying order appropriately until an attractor decomposition is obtained. This addition does not increase the worst-case asymptotic running time by more than a polynomial factor.

We have illustrated that our technique, when applied to the standard McNaughton-Zielonka algorithm, yields an algorithm that can solve some hard examples [13] in polynomial time. Other families of hard examples [32, 2] should also be analyzed. We believe that studying the shapes of attractor decompositions of hard examples and how they impact the intricate behaviour of symmetric attractor-based algorithms could shed new light on some central questions in the algorithmic study of parity games, such as how to overcome the quasi-polynomial barrier [6].

References

- 1 Massimo Benerecetti, Daniele Dell’Erba, and Fabio Mogavero. Solving parity games via priority promotion. *Formal Methods Syst. Des.*, 52(2):193–226, 2018. doi:10.1007/s10703-018-0315-1.
- 2 Massimo Benerecetti, Daniele Dell’Erba, and Fabio Mogavero. Robust worst cases for parity games algorithms. *Inf. Comput.*, 272:104501, 2020. doi:10.1016/j.ic.2019.104501.
- 3 Massimo Benerecetti, Daniele Dell’Erba, Fabio Mogavero, Sven Schewe, and Dominik Wojtczak. Priority promotion with parysian flair. *CoRR*, abs/2105.01738, 2021. arXiv:2105.01738.
- 4 J. C. Bradfield and I. Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.
- 5 Cristian S. Calude, Sanjay Jain, Bakhadyr Khossainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022. doi:10.1137/17M1145288.
- 6 W. Czerwiński, L. Daviaud, N. Fijalkow, M. Jurdziński, R. Lazić, and P. Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2333–2349. SIAM, 2019. doi:10.1137/1.9781611975482.142.
- 7 L. Daviaud, M. Jurdziński, and K. Lehtinen. Alternating weak automata from universal trees. In *30th International Conference on Concurrency Theory, CONCUR 2019, volume 140 of Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, Amsterdam, the Netherlands, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 8 Laure Daviaud, Marcin Jurdzinski, and K. S. Thejaswini. The strahler number of a parity game, 2020. doi:10.4230/LIPIcs.ICALP.2020.123.
- 9 Anuj Dawar and Erich Grädel. The descriptive complexity of parity games. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2008. doi:10.1007/978-3-540-87531-4_26.
- 10 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.
- 11 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ -calculus. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV ’93, Elounda, Greece, June 28 – July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 1993. doi:10.1007/3-540-56922-7_32.
- 12 Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 694–707. Springer, 2001. doi:10.1007/3-540-48224-5_57.

- 13 Oliver Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO Theor. Informatics Appl.*, 45(4):449–457, 2011. doi:10.1051/ita/2011124.
- 14 Maciej Gazda and Tim A. C. Willemse. Zielonka’s recursive algorithm: dull, weak and solitaire games and tighter bounds. In Gabriele Puppis and Tiziano Villa, editors, *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013, Borca di Cadore, Dolomites, Italy, 29-31th August 2013*, volume 119 of *EPTCS*, pages 7–20, 2013. doi:10.4204/EPTCS.119.4.
- 15 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 16 Ichiro Hasuo, Shunsuke Shimizu, and Corina Cirstea. Lattice-theoretic progress measures and coalgebraic model checking. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 – 22, 2016*, pages 718–732. ACM, 2016. doi:10.1145/2837614.2837673.
- 17 M. Jurdziński. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301, Lille, France, 2000. Springer. doi:10.1007/3-540-46541-3_24.
- 18 M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–9, Reykjavik, Iceland, 2017. IEEE Computer Society.
- 19 M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.
- 20 Marcin Jurdziński, Rémi Morvan, and K. S. Thejaswini. Universal Algorithms for Parity Games and Nested Fixpoints. *CoRR*, abs/2001.04333, 2020. arXiv:2001.04333.
- 21 J. J. A. Keiren. Benchmarks for parity games. In *FSEN*, volume 9392 of *LNCS*, pages 127–142, Tehran, Iran, 2015. Springer. doi:10.1007/978-3-319-24644-4_9.
- 22 Ruben Lapauw, Maurice Bruynooghe, and Marc Denecker. Improving parity game solvers with justifications. In *Verification, Model Checking, and Abstract Interpretation*, pages 449–470, Berlin, Heidelberg, 2020. Springer-Verlag. doi:10.1007/978-3-030-39322-9_21.
- 23 K. Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 639–648, Oxford, UK, 2018. IEEE. doi:10.1145/3209108.3209115.
- 24 K. Lehtinen, S. Schewe, and D. Wojtczak. Improving the complexity of Parys’ recursive algorithm, 2019. arXiv:1904.11810.
- 25 Karoliina Lehtinen, Paweł Parys, Sven Schewe, and Dominik Wojtczak. A Recursive Approach to Solving Parity Games in Quasipolynomial Time. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022. doi:10.46298/lmcs-18(1:8)2022.
- 26 R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. doi:10.1016/0168-0072(93)90036-D.
- 27 P. Parys. Parity games: Zielonka’s algorithm in quasi-polynomial time. In *MFCS 2019*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:13, Aachen, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2019.10.
- 28 N. Piterman. From nondeterministic buchi and streett automata to deterministic parity automata. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS’06)*, pages 255–264, 2006. doi:10.1109/LICS.2006.28.
- 29 Sven Schewe and Bernd Finkbeiner. Synthesis of asynchronous systems. In *Proceedings of the 16th International Conference on Logic-Based Program Synthesis and Transformation, LOPSTR’06*, pages 127–142, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/978-3-540-71410-1_10.

- 30 K. S. Thejaswini, Marcin Jurdziński, and Pierre Ohlmann. A technique to speed up symmetric attractor-based algorithms for parity games. *CoRR*, abs/2010.08288, 2020. [arXiv:2010.08288](https://arxiv.org/abs/2010.08288).
- 31 T. van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In *Tools and Algorithms for the Construction and Analysis of Systems, 24th International Conference, TACAS 2018*, volume 10805 of *LNCS*, pages 291–308, Thessaloniki, Greece, 2018. Springer. doi:10.1007/978-3-319-89960-2_16.
- 32 Tom van Dijk. A parity game tale of two counters. In Jérôme Leroux and Jean-François Raskin, editors, *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019*, volume 305 of *EPTCS*, pages 107–122, 2019. doi:10.4204/EPTCS.305.8.
- 33 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

A Set and Move operator

We define operations like “Set T^ϵ to S ”, “Add S to S_{Odd}^ω ” etc., as operations performed on sets rigorously. These short hands helped us capture the essence of these manipulations performed on the decompositions to give a new one. With appropriate data structures, these operations can be performed in nearly linear time, i.e, for a set of size k , it takes time at most $mk \log(n) \log(d)$.

Adding and removing vertices from $\mathcal{D}_{\text{Even}}^\epsilon$ and $\mathcal{D}_{\text{Odd}}^\omega$

To make the definition of Set and Move easier, we define operations that would facilitate this: First, we define \oplus and \ominus . We also define an extension of our notation $[\mathcal{D}_{\text{Even}}^\epsilon]$ here. We use $[[\mathcal{D}_{\text{Even}}^\epsilon]]$ to denote the set of vertices in $[\mathcal{D}_{\text{Even}}^\epsilon]$ along with S_{Even}^ϵ . We also introduce one extra partition of vertices to the already existing ones in the decompositions: \top_{Even} for the Even decomposition and \top_{Odd} for the Odd decomposition to accommodate vertices that no longer fit into a decomposition.

Consider an Even decomposition $\mathcal{D}_{\text{Even}}$ and an element of the Even tree ϵ at level h , and subset of vertices U , whose priorities are at most $h + 1$. Intuitively, we modify the decomposition $\mathcal{D}_{\text{Even}}^\epsilon$ by adding elements of a set U to it whilst maintaining all the properties of a decomposition. More rigorously, we let $[[\mathcal{D}_{\text{Even}}^\epsilon]] \leftarrow [[\mathcal{D}_{\text{Even}}^\epsilon]] \oplus U$ denote the following operations in that order:

- $S_{\text{Even}}^\epsilon \leftarrow S_{\text{Even}}^\epsilon \cup (U \cap \pi^{-1}(h + 1))$;
- $H_{\text{Even}}^\epsilon \leftarrow H_{\text{Even}}^\epsilon \cup (U \cap \pi^{-1}(h))$;
- $T_{\text{Even}}^\epsilon \leftarrow T_{\text{Even}}^\epsilon \cup (U \cap \pi^{-1}(< h))$.

One could analogously define the operators \oplus for $\mathcal{D}_{\text{Odd}}^\omega$ for an Odd decomposition.

To remove vertices U from the decomposition $\mathcal{D}_{\text{Even}}^\epsilon$, we define $[[\mathcal{D}_{\text{Even}}^\epsilon]] \leftarrow [[\mathcal{D}_{\text{Even}}^\epsilon]] \ominus U$ as follows: for each ϵ' in the subtree of ϵ ,

- $S_{\text{Even}}^{\epsilon'} \leftarrow S_{\text{Even}}^{\epsilon'} \setminus U$;
- $H_{\text{Even}}^{\epsilon'} \leftarrow H_{\text{Even}}^{\epsilon'} \setminus U$;
- $T_{\text{Even}}^{\epsilon'} \leftarrow T_{\text{Even}}^{\epsilon'} \setminus U$.

Setting T_{Even}^ϵ

To give an intuitive definition of “Set T_{Even}^ϵ to S ”, we essentially replace the set of vertices at T_{Even}^ϵ with S , and we remove vertices which are originally there which are not S and assign them to the next available positions in the decomposition that would be processed. For a subset of vertices S , we define “Set T_{Even}^ϵ to S ” where the set T_{Even}^ϵ contains S .

44:20 A Technique to Speed up Symmetric Attractor-Based Algorithms for Parity Games

- $R \leftarrow T_{\text{Even}}^\epsilon \setminus S$
- $T_{\text{Even}}^\epsilon \leftarrow S$
- $\llbracket \mathcal{D}_{\text{Even}}^{\epsilon_1} \rrbracket \leftarrow \llbracket \mathcal{D}_{\text{Even}}^{\epsilon_1} \rrbracket \oplus R$, where
 - ϵ_1 is the first child of ϵ in the tree if ϵ is not a leaf,
 - if ϵ is a leaf, then ϵ_1 is the smallest node larger than epsilon, and
 - if the smallest node larger than epsilon does not exist in the tree, we let an element $\top_{\text{Even}} \leftarrow R$.

Setting $S_{\text{Odd}}^{\omega_i}$

This is very similar to the above, the only difference is in the last line. We could unify these setting operations, but we give them separately for more clarity. For a subset S of vertices, we define “Set $S_{\text{Odd}}^{\omega_i}$ to S ” to be:

- $R \leftarrow S_{\text{Odd}}^{\omega_i} \setminus S$
- $S_{\text{Odd}}^{\omega_i} \leftarrow S$
- if i is the last child of ω , then $S_{\text{Odd}}^\omega \leftarrow S_{\text{Odd}}^\omega \cup R$
- if not, then $\llbracket \mathcal{D}_{\text{Odd}}^{\omega_{i+1}} \rrbracket \leftarrow \llbracket \mathcal{D}_{\text{Odd}}^{\omega_{i+1}} \rrbracket \oplus R$

Moving vertices to $S_{\text{Odd}}^{\omega_i}$

For a subset of vertices S , we define “Move S to at least $S_{\text{Odd}}^{\omega_i}$ ” as follows: we first remove S from all the other positions in the decomposition rooted at ω_i and then add it to $S_{\text{Odd}}^{\omega_i}$ if it was in $\llbracket \mathcal{D}_{\text{Odd}}^{\omega_i} \rrbracket$.

- $R \leftarrow \llbracket \mathcal{D}_{\text{Odd}}^{\omega_i} \rrbracket \cap S$
- $\llbracket \mathcal{D}_{\text{Odd}}^{\omega_i} \rrbracket \leftarrow \llbracket \mathcal{D}_{\text{Odd}}^{\omega_i} \rrbracket \ominus S$
- $S_{\text{Odd}}^{\omega_i} \leftarrow S_{\text{Odd}}^{\omega_i} \cup R$

The even counterpart is defined similarly.